Imitation Learning

Milind Nakul, Nakul Singh, Astitva Chaudhary

I. INTRODUCTION

In this term paper, we analyze various imitation learning algorithms and compare the performance between them. Imitation learning is a sequential prediction problem where future observations depend on previous actions taken. Hence it violates the common i.i.d. assumption made in various statistical learning problems. The aim is to replicate the behavior of the expert which has already been provided to us. It has applications in areas such as self-driving cars, robotics, and many more. To solve imitation learning problems various types of algorithms have been proposed using different techniques. Some of the popular algorithms include Behavioral Cloning, Stochastic Mixing Iterative Learning Algorithm, Forward Training Algorithm, and Dataset Aggregation Algorithm. We compare the performances of two of these algorithms against the Behavioral Cloning algorithm which simply learns a supervised learning classifier to replicate the behavior of the expert policy provided.

II. ALGORITHMS

A. Behavioral Cloning

This is one the traditional methods used to solve imitation learning problems. It completely ignores the dependence of the current state on the previous predictions made and simply trains the agent on the states encountered by the expert policy. The objective function that this algorithm optimizes is as follows:

$$\hat{\pi}_{sup} = argmin_{\pi \in \Pi} E_{s \sim d^*_\pi}[l(s, \pi)] \tag{1}$$

Here $l(s, \pi)$ represents the loss function of taking an action which is different expert action. We can see that this algorithm only optimizes under states induced by the expert policy and hence performs badly in cases where states outside the known distribution are encountered. It has been shown in literature that the loss for this algorithm grows quadratically in event horizon T. The event horizon is the duration of one task which has been assigned to the agent(More on this in theoretical analysis).

Intuitively, the reason that the loss function is quadratic in T is that as soon as the algorithm makes a mistake in any one state, it continues to encounter states outside the distribution which was induced by the expert policy and hence the mistake gets compounded. This can be seen as the problem of out of distribution testing which is encountered in alot of machine learning problems, where the distribution on which the agent is tested is different from the one on which it was trained.

B. Forward Training Algorithm

The basic idea behind Forward Training algorithm is to change the policies slowly so that the learner is given enough time to recover from mistakes. This trains a deterministic but non-stationary policy. We define a new policy for each time step. The implementation of this is very similar to that of the SMILe algorithm. A drawback of the forward algorithm is that it is impractical when T is large (or undefined) as we must train T different policies sequentially and cannot stop the algorithm before we complete all T iterations.

The implementation of this algorithm is not done since the code is very similar in nature to the SMILe algorithm. The only difference being that instead of stochastically mixing of algorithms as done in SMILe, this algorithm just trains a new policy at each iteration.

C. Stochastic Mixing Iterative Learning

The problem with behavioral cloning is that it fails to recover from the mistakes it has made because of the out of distribution states it encounters after making a mistake. The SMILe algorithm aims to solve this problem by training a stationary stochastic policy over several iterations. At the start of each iteration a new policy is obtained by stochastically mixing the current policy with the expert policy. This mixing is done in such a way so as to ensure that the contribution of the expert policy reduces, as the number of iterations increase and for $N \to \infty$, the expert policy is completely removed from the final policy learned.

Algorithm 1 SMILe Algorithm

Initialize $\pi^0 \to \pi^*$
while $i \leq N$ **do**
  Sample a T step trajectory using the current policy
  Get the dataset $D = (s, \pi^*(s))$ using the expert policy
  Train a classifier on the data collected $\pi^{*i} = argmin_{\pi \in \Pi} E_{s \sim D}[e(s, \pi)]$
  Stochastic Mixing : $\pi^i = (1-\alpha)^i \pi^* + \alpha \sum_{j=1}^{i} (1-\alpha)^{j-1} \pi^{*j}$
  Train a policy on the entire dataset D
end while
Remove the expert policy completely $\tilde{\pi^N} = \frac{\pi^N - (1-\alpha)^N \pi*}{1-(1-\alpha)^N}$

Return the final trained policy π

Our implementation of SMILe is slightly different from the one that is provided in the paper. We use neural networks as function approximators for the stochastic policy, hence the direct stochastic mixing of policies is very difficult. We

initialize N different policies and we have weights associated with each policy. At the start we execute the expert policy and collect some dataset. Then a policy is selected from the set of policies and is trained on the current dataset. The weight of each policy is updated based on the number of times it has been trained and the current iteration. Then at the next iteration we select a new policy based on the weights to generate the new dataset and the process is repeated. Finally we return the best performing policy.

D. Dataset Aggregation

This is an iterative procedure to train a deterministic policy for the imitation learning problems. It provides us with good convergence guarantees under the distribution of states it induces.

The basic idea for this algorithm is to make it robust to out of distribution states which might be encountered by the agent. It starts by training a classifier on the states and actions of the expert policy. Then this trained policy is used to gather a new set of states that might be encountered. Then the expert is queried on the states regarding the actions that it would on these newly encountered states and they are aggregated together with the entire dataset.Then a new policy is trained on the entire dataset. These steps are repeated until we converge to a deterministic policy. Hence over many iterations we train our agent over states that it is likely to encounter during execution hence making it less likely that out of distribution states are encountered by the agent.

Algorithm 2 DAGGER

Ensure: $D \rightarrow \phi$
  $\pi \rightarrow \pi^*$
  while $i \leq N$ **do**
    Sample a T step trajectory using the current policy
    Get the dataset $D_i = (s, \pi^*(s))$ using the expert policy

    Aggregation step : $D \leftarrow D \cup D_i$
    Train a policy on the entire dataset D
  end while
  Return the final trained policy π

DAGGER proceeds by collecting a dataset at each iteration under the current policy and trains the next policy under the aggregate of all collected datasets. It provides us with a linear bound on the loss function in the event horizon T.This is because the policy is trained on a dataset which contains states from outside the distribution of states induced by just the expert policy.

III. Theoretical Analysis

A. Notation

π^* represents the expert policy we wish to emulate. T refers to the task horizon and it is the number of time steps of one episode of a task. $C(s,a)\ in\ [0,1]$ denotes the immediate cost of performing an action a in state s. Equivalently $E[C_\pi(s,a)]$ denotes the expected cost of performing a policy π in the state s.

$e(s,a) = I(a \neq \pi^*(s))$, is the 0-1 loss of performing an action in state s as compared to the expert policy $\pi^*(s)$. $e_\pi(s,a) = E_{a\sim\pi}[e(s,a)]$ is the expected 0-1 loss of policy π in state s.

d^i_π denotes the distribution of states at time step i under the policy π. The distribution of states over T time steps is denotes as $d_\pi = \frac{1}{T}\sum_{i=1}^{T} d^i_\pi$. The expected T-step cost of executing a policy π is denoted as $J(\pi) = TE_{s\sim d_\pi}[C_\pi(s)]$. The regret of a policy under a particular policy class is denoted as $R_\Pi(\pi) = J(\pi) - min_{\pi'\in\Pi} J(\pi')$.

B. Behavioral Cloning

We have the following convergence guarantee for Behavioral Cloning algorithm[2]:

Let $\hat{\pi}$ be such that the probability of making a mistake under d_{π^*} is ϵ, i.e. $E_{s\sim d_{\pi^*}}[e_{\hat{\pi}}(s)] \leq \epsilon$. Then $J(\hat{\pi}) \leq J(\pi^*) + T^2\epsilon$.

Proof: $J(\hat{\pi}) \leq \sum_{t=1}^{T}[p_{t-1}E_{s\sim d_t}(C_{\hat{\pi}}(s)) + (1-p_{t-1})] \leq \sum_{t=1}^{T}[p_{t-1}E_{s\sim d_t}(C_{\pi^*}(s)) + p_{t-1}e_t + (1-p_{t-1})] \leq J(\pi^*) + \sum_{t=1}^{T}\sum_{i=1}^{t}\epsilon_i \leq J(\pi^*) + T\sum_{t=1}^{T}\epsilon_t = J(\pi^*) + T^2\epsilon$

Here p_t is the probability that $\hat{\pi}$ hasn't made a mistake under the distribution $d^t_{\pi^*}$. d_t is the distribution of states faced by $\hat{\pi}$ at time step t, conditioned on the fact that no mistakes have been made so far. Similarly $d_t^{'}$ represents the distribution of states faced by $\hat{\pi}$ at time step t, conditioned on the fact that at least one mistake has been made so far. ϵ_t represents the probability that $\hat{\pi}$ makes a mistake under the distribution $d^t_{\pi^*}$. We can write $\epsilon = \frac{1}{T}\sum_{t-1}^{T}\epsilon_t$

The first inequality follows from the fact that if a mistake has been made the maximum values of the expected immediate cost will be 1. Let e_t and $e_t^{'}$ represent the probability of $\hat{\pi}$ making a mistake under the distributions d_t and $d_t^{'}$. We can say that $E_{s\sim d_t}[C_{\hat{\pi}}(s)] \leq E_{s\sim d_t}[C_{\pi^*}(s)] + e_t$.This is just using the fact that maximum value of the expected immediate loss will be less than the expected immediate loss of the expert policy plus e_t since maximum $C(s,a)$ is bounded in $[0,1]$. The second inequality follows from this. From the definitions above we can write the T step cost of the expert policy as, $J(\pi^*) = \sum_{t=1}^{T} p_{t-1}E_{s\sim d_t}[C_{\pi^*}(s)] + (1-p_{t-1})E_{s\sim d_t^{'}}[C_{\pi^*}(s)]$. Hence $\sum_{t=1}^{T} p_{t-1}E_{s\sim d_t}[C_{\pi^*}(s)] \leq J(\pi^*)$. We can also write $\epsilon_t = p_{t-1}e_t + (1-p_{t-1})e_t^{'}$ The third inequality follows from these.

C. Stochastic Mixing Iterative Learning Algorithm

The SMILe algorithm provides us with the following convergence guarantees[2]:

Let the expected T-step cost of executing $\pi\prime$ at the steps $\{t_1, \cdots, t_k\}$ be $J^\pi_k(\pi\prime, t_1, \cdots, t_k)$. Let the expected T-step cost of executing $\pi\prime$ k times and policy $\frac{1}{\binom{T}{k}}\sum_{t_1=1}^{T-k+1}\cdots\sum_{t_k=t_{k-1}+1}^{T} J^\pi_k(\pi\prime, t_1, \cdots, t_k)$. Let the k^{th} order policy disadvantage of $\pi\prime$ with respect to π be $A_k(\pi, \pi\prime) = \bar{J}^\pi_K(\pi\prime) - J(\pi)$. α is the small probability of executing the new policy $\hat{\pi}^{n+1}$. We can then have, if $\alpha \leq \frac{1}{T}$ and $\forall$ k $\in \{1, 2, \cdots, T-1\}$, $J(\pi^n) \leq J(\pi^0) + n\sum_{i=1}^{k}\alpha^i\binom{T}{i}(1-\alpha)^{T-i}\bar{A}_i + n\alpha^{k+1}T\binom{T}{k+1}$ where, $\bar{A}_i =$

$\frac{1}{n}\sum_{j=1}^{n} A_i(\pi^{j-1}, \tilde{\pi}^j)$. On every time step, π^n queries the expert π^0 with probability $p_n = (1-\alpha)^n$.

Let π^0 be the expert policy, and $\tilde{\pi}^n$ be the unsupervised policy which never queries the expert. We can bind the performance of $\tilde{\pi}^n$ as $J(\tilde{\pi}^n) \leq J(\pi^n) + p_n T^2$. [2]

Proof[4]: $J(\pi^n) \geq (1-p_n)^T J(\tilde{\pi}^n) + p_n T(1-p_n)^{T-1} \bar{J}_1^{\tilde{\pi}^n}(\pi^0) \geq J(\tilde{\pi}^n) + p_n T[(1-p_n)^{T-1} \bar{J}_1^{\tilde{\pi}^n}(\pi^0) - J(\tilde{\pi}^n)] \geq J(\tilde{\pi}^n) - p_n T^2$

Here, $p_n = (1-\alpha)^n$, where α is the small probability of executing the new policy $\hat{\pi}^{n+1}$. An important implication of this lemma is that when $n \geq \frac{2}{\alpha}\ln T$, it implies that $p_n \leq \frac{1}{T^2}$ which implies that $J(\tilde{\pi}^n) \leq J(\pi^n) + 1$, where the additional cost of 1 becomes insignificant for large T, so it can be ignored. π^n never queries the expert with probability $(1-p_n)^T$. It has T-step cost of $J(\tilde{\pi}^n)$. At time t, π^n queries the expert once with probability $p_n(1-p_n)^{T-1}$. It has T-step cost of $J_1^{\tilde{\pi}^n}(\pi^0, T)$.

For SMILe we also have the following guarantee[2]: when $\alpha = \frac{\sqrt{3}}{T^2\sqrt{\log T}}$ and $N = 2T^2(\ln T)^{3/2}$ then $J(\tilde{\pi}^N) \leq J(\pi^*) + O(T(\tilde{A}_1 + \tilde{\epsilon}) + 1)$

Here, $\epsilon_i = E_{s\sim d_{\pi^{i-1}}}(e(s, \hat{\pi}^{*i}))$, $\tilde{\epsilon} = \frac{\alpha}{(1-(1-\alpha)^n)}\sum_{i=1}^{n}(1-\alpha)^{i-1}\epsilon_i$, $\tilde{A}_1 = \frac{\alpha}{(1-(1-\alpha)^n)}\sum_{i=1}^{n}(1-\alpha)^{i-1} A_1(\pi^{i-1}, \hat{\pi}^{*i} \mid \pi^*)$ and $A_1(\pi^{i-1}, \hat{\pi}^{*i} \mid \pi^*) = \bar{J}_1^{\pi^{i-1}}(\hat{\pi}^{*i}) - \bar{J}_1^{\pi^{i-1}}(\pi^*)$. Usually, $\tilde{A}_1$ will be at least of the order of $\tilde{\epsilon}$, then $O(T(\tilde{A}_1 + \tilde{\epsilon}) + 1)$ can be written as $O(T\tilde{A}_1 + 1)$. So, SMILe takes $O(T^2(\ln T)^{3/2})$ steps to guarantee a regret of $O(T\tilde{A}_1 + 1)$.

D. Forward Training Algorithm

We have the following relation for expected T-step cost of policy π^n and the policy disadvantage A. $A(\pi^{i-1}, \pi_i^i) = J^{\pi^{i-1}}(\pi_i^i, i) - J(\pi^{i-1})$. Policy disadvantage can be intuitively understood as the increase in J when we change the current policies at the i^{th} step, keeping it same elsewhere. With the above definition of policy disadvantage we have $J(\pi^n) = J(\pi^*) + n\bar{A}$ where $\bar{A} = \frac{1}{n}\sum_{i=1}^{n} A(\pi^{i-1}, \pi_i^i)$.
Proof : In Fast forward training in the i^{th} iteration we only update the policy π_i^i keeping the policies at other steps same as the policies at those steps in the previous iteration i.e $\pi_j^i = \pi_j^{i-1} for j \neq i$. With this policy update procedure we see that the T-step cost($J(\pi^i)$) at the i^{th} iteration can be obtained from $J(\pi^i) = J^{\pi^{i-1}}(\pi_i^i, i) = J(\pi^{i-1}) + A(\pi^{i-1}, \pi_i^i)$. Solving this recurrence relation gives the results described in theorem 3.1 given in [2].

E. Dataset Aggregation

The DAGGER algorithm provides us with the following convergence guarantees[3]:

If number of iterations of DAGGER is $\hat{O}(T)$, then there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ such that $E_{s\sim d_{\hat{\pi}}}[l(s, \hat{\pi})] \leq \epsilon_N + O(1/T)$

If number of iterations of DAGGER is $\hat{O}(uT)$, then there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ such that $J(\hat{\pi}) \leq J(\pi^*) + uT\epsilon_N + O(1))$

Hence the DAGGER algorithm guarantees that the expected loss under the best policy grows linearly in the event horizon T, and provides more robust policies as compared to Behavioral Cloning and SMILe algorithms.

IV. Simulation

A. Proximal Policy Optimization(PPO)

PPO is one of the state of the art policy gradient methods in Reinforcement Learning. It directly optimizes the parameters of a policy using the gradients. The main advantage of the PPO is that it uses a clipped objective function which reduces the variance of the earlier policy gradient methods and makes it more sample efficient. We trained our expert using the PPO algorithm on the MountainCar and CartPole environment.

For the implementation of PPO we have used the code available on the keras website(link)

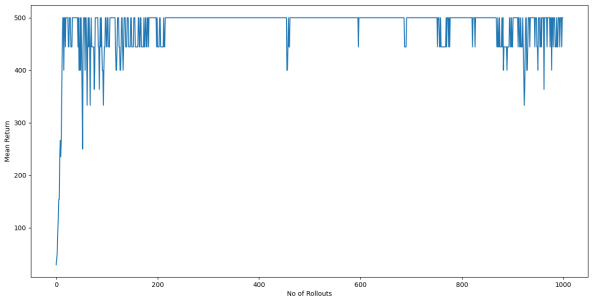

Fig. 1. CartPole Environment

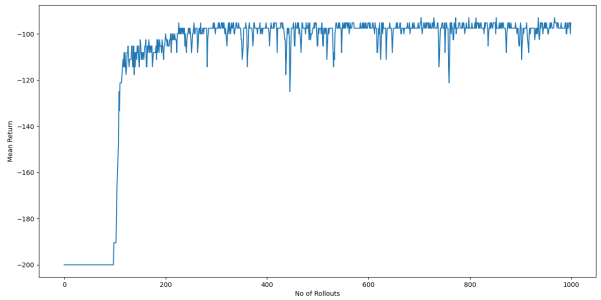

Fig. 2. MountainCar Environment

Fig. 3. Expert Training results using PPO

We can see from the simulation results that the MountainCar environment is more difficult to solve than the CartPole environment. This is attributed to the sparse reward space of the MountainCar environment. The PPO agent for the CartPole environment is able to solve it in about 20 rollouts, on the other hand it requires around 100 rollouts for the PPO agent to solve the MountainCar environment.

B. Environment Description

- **CartPole Environment** : On of the most basic classic control environments in OpenGym AI. A pole is attached by an unactuated joint to a cart, which moves along a friction-less track. The objective is to keep the

pole upright as long as possible by either moving the cart to the left or right. There is a *+1* reward for keeping the pole upright in each time-step and one episode has a maximum of 500 time-steps.

- **MountainCar Environment** : This a slightly difficult environment to solve because of its sparse reward space. A car is on a one-dimensional track, positioned between two ”mountains”. The goal is to drive up the mountain on the right. Reaching the top of the mountain has a *+1* reward and the other time steps have *-1* reward. We need to build momentum for the car using three actions left, right or stay still.

C. Results for CartPole Environment

Fig 4 shows the plot of No of Rollouts vs the mean return over each rollout for the Behavioral Cloning algorithm. The total number of rollouts is 10. Each episode of of the CartPole environment runs for 500 steps. Hence total number of state-action pairs over which our algorithm is trained is 5000.

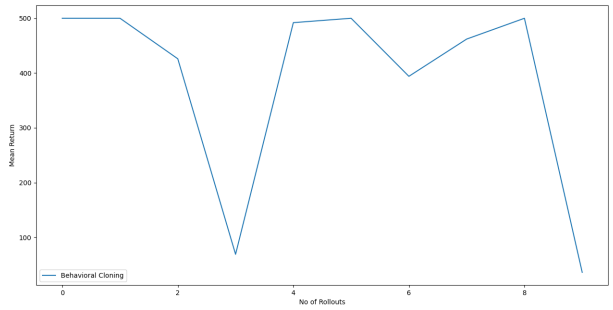

Fig. 4. CartPole Environment

Fig 5 shows the plot of No of Dagger iterations vs Mean Return. For each dagger iteration we have 10 rollouts and hence the total number of samples over which the agent is trained is 50000.

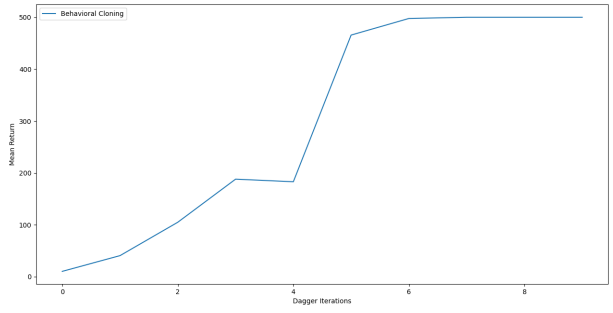

Fig. 5. CartPole Environment

Fig 6 and Fig7 shows the plot of No of Rollouts vs the Mean Return for the SMILe algorithm for different policies that are trained. Each curve in the plot is obtained by training the stochastically selected policy identified by it’s index in the list. It converges in about 25 rollouts and we have a total of 15 SMILe iterations. Hence the total number of samples required is about 75000.

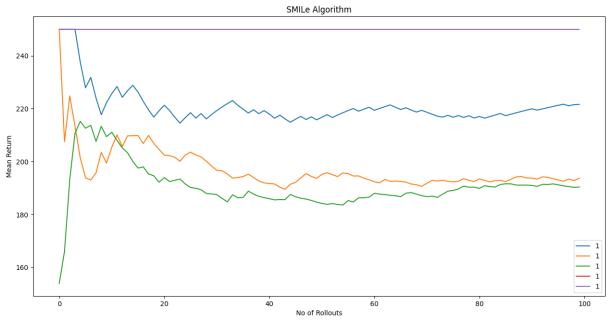

Fig. 6. CartPole Environment

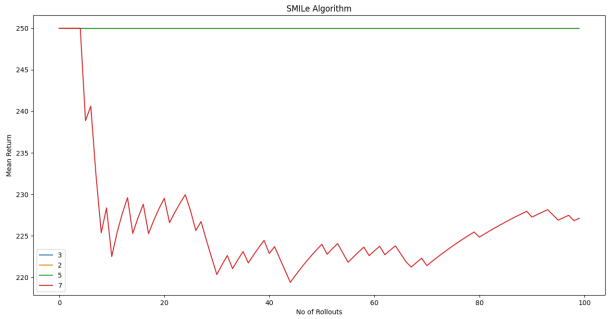

Fig. 7. CartPole Environment

D. Results for MountainCar Environment

Fig 8 is the plot of Episodes vs the Return of each episode. We have 10 dagger iterations and 10 rollouts and hence the total number of episodes is 100. Total samples is around 20000 since each episode has a maximum of 200 time steps.

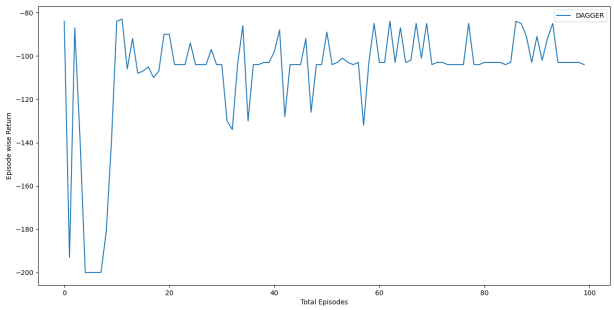

Fig. 8. MountainCar Environment

Fig 9 shows the mean return over the number of DAGGER iterations.

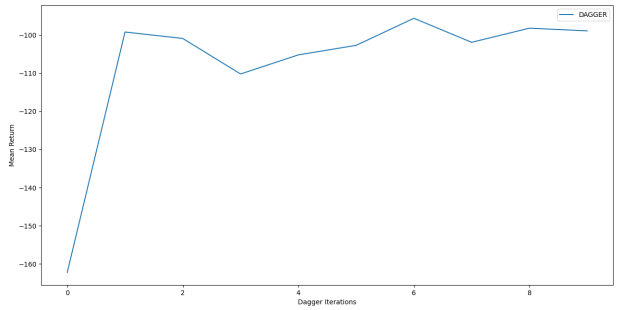

Fig. 9. MountainCar Environment

Fig 10 shows the return over the number of rollouts for the Behavioral Cloning algorithm.

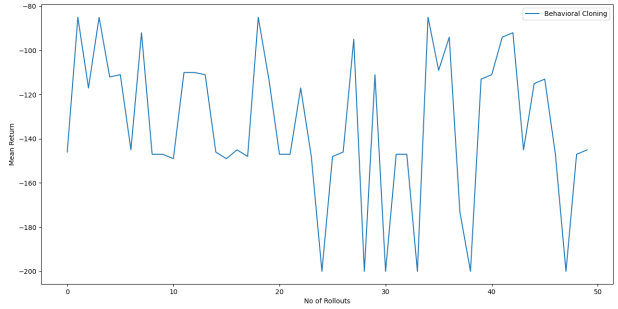

Fig. 10. MountainCar Environment

Fig 11 and Fig 12 shows the plot of No of Rollouts vs the Mean Return for the SMILe algorithm for different policies that are trained. The index notation is same as described for the SMILe plots of CartPole environment. It converges after about 20 rollouts. Hence the total samples required is 60000.

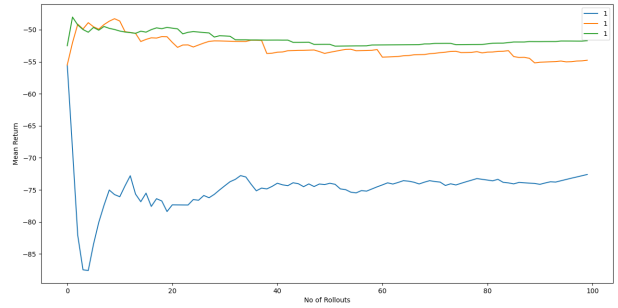

Fig. 11. MountainCar Environment

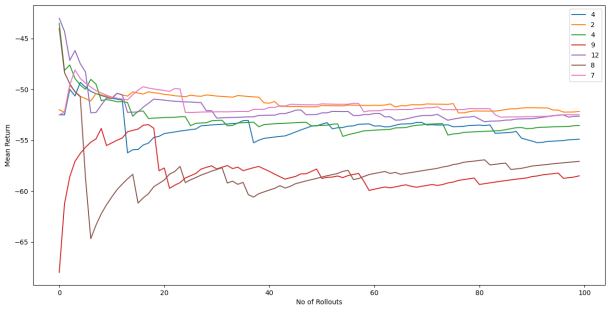

Fig. 12. MountainCar Environment

Algorithm	Parameter	value
SMILe	N	15
	α	0.1
	Iterations	15
	Roll-outs	100
DAGGER	β_1	1
	$\beta_j (j > 1)$	0
	Iterations	10
	Roll-outs	10
Behavioral cloning	Iterations	1
	Roll-outs	10

In DAGGER algorithm, we take $\beta_1 = 1$ and rest of βs to be zero because at the first iteration we generate the policy using the expert policy and after that we don't need to use it in data generation since we collect the data using the already trained policy.

E. Observations

1) CartPole Environment: We have plotted the mean reward against the number of rollouts of the environment that were made for the CartPole environment. Comparing Fig.4 and Fig.5, we can see that the DAGGER algorithm provides a much more robust agent as compared to the Behavioral Cloning algorithm. It reaches a mean reward of around 500 after about 6 rollouts and once the agent has solved the environment the average reward does not dip. However, in case of Behavioral Cloning we see that it is able to learn much quicker as compared to DAGGER, the average reward dips for certain rollouts and continues to decrease. This can be attributed to the fact that once the agent makes a mistake, it encounters states outside the one induced by the expert policy and hence the number of mistakes grow further. This results in the lack of robustness of the Behavioral Cloning algorithm as compared to DAGGER algorithm.

For the SMILe simulation results, we have considered a list of 15 policies and $\alpha = 0.1$. For each iteration, first we randomly select a policy and then train it over the dataset generated from the previous iteration. This sampling of policies is done such that as the number of iterations increase the chances of getting the expert policy decreases

exponentially. Thus for N=15 we train our modelled policies for 9 iterations.
As can be seen from Fig 6, policy indexed at 1 is sampled the most and hence eventually reaches the optimal mean return possible.
The policies in Fig 7 (indexed at 2,3,5,7) are those which are sampled after the policy indexed at 1 has been trained at least once. Therefore the optimal mean return for these policies are achieved in a smaller number of roll outs. Compared to Behavioural cloning algorithm the SMILe algorithm provides more robustness and a larger optimal mean value as it changes the policies stochastically allowing for the learner to recover from mistakes.Compared to DAGGER, SMILe gives inferior results because of it's stochastic nature.

2) MountainCar Environment: In the case of the MountainCar environment as well, a similar trend is seen. The average reward plot for the DAGGER algorithm has a lot less variance after a certain number of DAGGER iterations have been completed. But the Behavioral Cloning algorithm has a lot more variance due to the fact that it is less robust against out of distribution states encountered by the agent.

We have also plotted the Reward against the number of episodes plot to get an idea about the sample efficiency of the algorithms. In case of Behavioral Cloning, this plot is same as the plot against number of rollouts since there is no need for data aggregation and training again. For the MountainCar environment, the DAGGER is able to converge to a good enough policy in about 20 episodes. The behavioral cloning algorithm outperforms the DAGGER algorithm at the starting episodes, however once the DAGGER converges it continuously outperforms Behavioral Cloning due to the better robustness of DAGGER.

For SMILe, compared to the CartPole environment here also we see similar results. The policy indexed at 1 is trained in the first iteration (denoted by the blue line in Fig 11) after that the rest of the policies take lesser number of roll-outs to converge to an optimal mean value.

In terms of sample efficiency we see that Behavioral Cloning algorithm is best since it requires a lot less number of samples to get trained. DAGGER performs better than SMILe for both environments in case of sample efficiency.

V. FUTURE WORK

For the SMILe algorithm our implementation is slightly different from that provided by the authors. To implement the loss function using Tensorflow functionalities we made several simplifications. Also the optimal mean value attained by the SMILe algorithm in case of MountainCar environment is better than the expert trained on PPO optimization. This is because we could not remove the expert policies completely as given in [2]. An efficient implementation for this process can be developed to get more accurate results.
The comparison for the considered algorithms can be extended to environments in the MuJoCo package as they are closely related to robotics applications.

All the algorithms we have tested assumed that the expert policy can be queried at any time and an unlimited number of times. However, querying the expert policy is a very costly operation. Hence we can look to implement algorithms which limit the number of expert queries as much as possible. This will help us in significantly reducing the execution time of the implementation of these algorithms.

REFERENCES

[1] Takayuki Osa; Joni Pajarinen; Gerhard Neumann; J. Andrew Bagnell; Pieter Abbeel; Jan Peters, An Algorithmic Perspective on Imitation Learning , now, 2018.
[2] S. Ross and J. A. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.
[3] Ross, Stéphane et al. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." AISTATS (2011).
[4] S. Ross and J. A. Bagnell. Efficient Reductions for Imitation Learning Supplementary Material.
[5] https://keras.io/examples/rl/ppo_cartpole
[6] https://github.com/jj-zhu/jadagger