Efficient Deep Reinforcement Learning with Imitation

Yuanxun Shao, Haoxin Ma, Haoliang Jiang
Georgia Institute of Technology

Abstract

Making optimal control of robots under dynamic and stochastic environments is indeed challenging due to the high algorithm. In this project, we attempt to demonstrate the efficiency of Learning from Demonstration (LfD) by comparing the effectiveness of three learning algorithms armed with LfD over our baseline that is a deep Q-learning algorithm with human demonstrations as a warm start. We tried three new directions: Deep TAMER, A2C with refined update directions, and Generative Adversarial Imitation Learning. Our environment is LunarLander-v2 from OpenAI Gym.

1. Introduction

To study how LfD improves the efficiency of reinforcement learning (RL), we attempt to develop three approaches and analyze their performances.

Our environment is the LunarLander-v2 game built in the OpenAI gym[1]. The task is to land the agent in the landing pad with zero velocity. The available actions are to activate the main engine pushing the agent up, the left engine pushing right, the right engine pushing left, or do nothing. The state space of this task is continuous with 8 components $[x, y, u, v, \alpha, \omega, L, R]$,representing correspondingly the position linear velocity, tilt angle and angular velocity of the agent, and whether the left (L) and right (R) leg touches the ground. During the game, for each frame a shaping factor is defined as $s = -100 \times (\sqrt{x^2 + y^2} + \times\sqrt{u^2 + v^2} + |\alpha|) + 10 \times (L + R)$, and the basic reward for frame k is $r_{kb} = s_k - s_{k-1}$. Also, actions will result in punishment r_{kp}, which is -0.3 for main engine and -0.03 for left or right engine. If the agent crushes or successfully lands, a -100 or $+100$ reward will be received. The task is demonstrated in Figure 1. The task is considered as solved when the average reward is above 200 for 100 consecutive episodes. And the performance of different algorithms is compared based on how many episodes the agent needs to train before the task is solved.

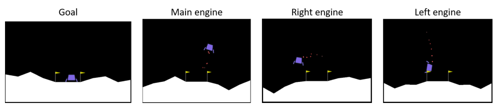

Figure 1. A demonstration of the task

Reward shaping is one of the techniques that enable the agent to learn from humans in a natural way of communication. And also, learning a reward function directly from human feedback, and optimize it using reinforcement learning is an effective and efficient way to improve on imitation learning[2]. In this project, we combined Deep TAMER algorithm[3] with DQN and improved upon this algorithm by calculating a discount factor for the current state, and discount the predicted human reward by this factor in order to reduce the affect of conflicting or wrong human feedback.

Memory replay is another key technique behind many recent advances in deep RL. Allowing the agent to learn from earlier memories can speed up learning and break undesirable temporal correlations [4]. To make the best use of human demonstrations and experiences, memory replay can recursively exploit them and take advantages of high computational power. In the baseline, DQN + LfD + memory replay is about 20x faster than just DQN + LfD without memory replay. Despite memory replay dramatically boosts RL performances [5], very little is understood about the basic properties of memory replay, especially combining with human demonstrations.

Generative adversarial methods have been shown as promising learning methods for generative tasks. The GANs transfer the training of a generative network to a two-player minimax game where a generator G is trained to learn a distribution f with a discriminator D[6]. Among many related work, J. Ho and S. Ermon [7] proposed a generative adversarial imitation learning (GAIL) method to combine reinforcement learning algorithms and inverse reinforcement learning algorithms by drawing an analogy between imitation learning and generative adversarial networks. Though GAIL achieved excellent performances in many tasks, no research has focus on its learning speed and performance in tasks requiring long time sequence to complete. In this project, we will study how the framework functions under our environment and how fast a naive GAIL can learn compared to other reinforcement learning and imitation learning methods.

The related works are described in section 2. The implemented and improved algorithms are illustrated in details in section 3. Section 4 includes details about the experiments. In section 5 and 6, we present the results and draw the conclusions.

2. Related Works

2.1. TAMER

There have been a large number of researches on learning from human ratings or rankings, and the deep TAMER framework[3] is a deep learning extension of TAMER[8], which is a framework trying to learn a human reward function from the positive and negative feedbacks received from human teachers. The deep TAMER framework takes advantage of deep learning to learn tasks with a complex state space. Also, the TAMER and deep TAMER frameworks have been proven to be very effective when combined with RL algorithms such as SARSA[9, 10]. Inspired by these previous works, we combined deep TAMER framework and DQN in a way of *action bias*[9], where deep TAMER and DQN are trained separately, and when selecting actions, these two predicted rewards are combined, and the action that maximize the combined reward is selected in ϵ-greedy way.

$$\tilde{V}(s) = \tilde{Q}(s) + \beta \tilde{H}(\overrightarrow{f}(s,a)) \quad (1)$$

where s stands for the current state, a stands for actions, $\overrightarrow{f}$ is a feature vector calculated from state-action pair (s, a), $\tilde{Q}$ stands for the predicted environment reward, $\tilde{H}$ stands for the predicted human reward, β is a factor which is calculated from up-till-now eligibility matrix and current feature vector $\overrightarrow{f}$, and $\tilde{V}$ is the combined reward. Details about β can be found in [9].

However, during the training, we found a problem that the TAMER + DQN algorithm is not performing better than DQN only. We think this is because there are conflicting and even wrong ones in human feedbacks. Conflicting or wrong feedbacks are possible or even inevitable since in this task of LunarLander, there are many ambiguous states where the human teacher may not know which action is the best or even the better one due to the large state space and limited action space. To validate this thought, we conducted an experiment where we generated an oracle using a trained agent (considered as expert, which can give consistent and correct feedback all the time), and trained new agents with the oracle and human teacher separately. The results are shown in Figure 2 and Table 1.

Based on the results, it is clear that the poor performance is due to conflicting and wrong feedbacks from human teachers, because while agent trained with human feedbacks performs poorly, agent trained with oracle, which doesn't include any conflicting or wrong feedback, obviously outperforms the baseline and DQN. In the following section 3.1, we will introduce our effort to address this problem and improve upon deep TAMER + DQN.

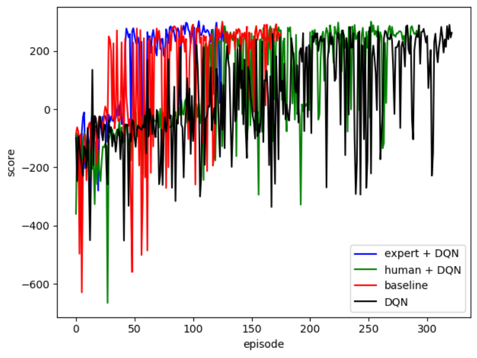

Figure 2. Comparison between expert, human, baseline, and DQN

Algorithms	Expert+DQN	Human+DQN	Baseline	DQN
Episode	132	293	176	322

Table 1. Comparison between expert, human, baseline, and DQN

2.2. On-Policy RL Memory Replay

The memory replay technique has been widely implemented in RL experiments currently and is shown to have a good performance [4, 5]. The concept of reusing human demonstrations is evolved from recomputing previous experience by means of dynamic programming. Application of the importance sampling technique made it possible to optimize a parametric control policy with the use of experience gained with other policies [11]. Particularly, memory replay is good for off-policy RL algorithms like DQN [4]. Because in off-policy RL algorithms, human demonstrations are generated by the same policy in expectation. Thus we can do non-sequential replay to efficiently exploit demonstrations. As shown in our baseline, this single technique made the algorithm 20x faster. But in terms of on-policy RL, e.g. A2C, the basic properties of memory replay is not well understood. The main challenge is the human demonstrations generated under different policies. The discrepancy between current policy and previous policies could lead to bad performance.

2.3. Generative Adversarial Imitation Learning

Generative adversarial learning methods become a popular research topic from the propose of GAN [6]. GANs transfer the training of a generative network to a two-player minimax game where a generator G is trained to learn a distribution f with a discriminator D. Built upon GAN, conditional GAN [12] develops a method to control the mapping from input to output by conditioning the standard generator G and discriminator D on extra information. The loss function is shown as following,

$$\begin{aligned} min_G max_D V(G, D) = E_{x,y}[log D(x, y)] + \\ E_x[log(1 - D(x, G(x)))] \end{aligned} \quad (2)$$

In all variant versions of generative adversarial networks, GAIL focus on robotic learning area. It trains a reinforcement learning algorithm and a inverse reinforcement learning algorithm at the same time to extract a policy from data directly. N. Baram et al. propose a model-based GAIL, dubbed MGAIL, which benefits the training of policy network by using extra gradient of the discriminator [13]. J. Fu et al. propose AIRL which is a practical land scaleable inverse reinforcement learning algorithm based on adversarial reward learning formulation [14]. To avoid environment interactions in imitation learning algorithms, I. Kostrikov et al. propose Discriminator-Actor-Critic which demonstrates effective performance in dealing with absorbing states [15]. Besides learning from demonstration, imitation from observation has also be explored by generative adversarial imitation learning models proposed by F. Torabi et al[16].

3. Algorithms

3.1. TAMER

In the previous section 2, we have shown that the poor performance of the deep TAMER + DQN algorithm is due to the conflicting and wrong feedbacks from human teachers. To solve this problem, a memeber of our group has proposed to discount the predicted human reward by a calculated factor γ_c, which measures how trustworthy the human feedbacks are, and tried three different methods to implement that.

- **Gaussian Mixture Model (GMM)**

  The first method is to use a GMM to classify all the past feedbacks into clusters, and update the model every time a new feedback is received. The discount factor γ_c for GMM consists of two parts, the log likelihood p of the GMM model itself, and how consistent the human feedbacks of the states which belong to the same cluster as the current state are. The consistency of cluster i is calculated as $c^i = \frac{max(n^i_{pos}, n^i_{neg})}{n^i_{total}}$, where n^i_{pos}, n^i_{neg} stand for the number of positive and negative rewards in cluster i separately, and n^i_{total} stands for the total number of feedbacks in cluster i. Then, the discount factor γ_c is calculated as

  $$\gamma_c = p \cdot c^i \tag{3}$$

- **Confidence**

  The second method is to ask the human teachers to provide both positive/negative reward, and how confident they are about this feedback at each time frame in the form of a confidence value c during training. To simplify the problem and reduce the thinking time of the human teachers, only two options of "confident" $(c = 0.7)$ or "not confident" $(c = 0.3)$ are available. Then, a neural network is trained to predict the confidence value $\tilde{c}(s)$ for the current state s. The discount factor $\gamma_c = \tilde{c}(s)$.

- **Bayesian**

  The third method is to replace the linear regression layer of deep TAMER with a Bayesian linear regression layer, so that the network will learn to predict a distribution (μ_a, σ_a) for each action a. Then a sample is drawn from the distribution for each state as predicted human rewards. In this method, $\gamma_c \equiv 1$. But since a Bayesian linear regression is used, the trustworthiness of the human feedbacks are already considered in the form of reward distribution.

When selecting actions, Equation 1 is modified as follows, and the action maximize $\tilde{V}(s)$ will be chosen in ϵ-greedy way.

$$\tilde{V}(s) = \tilde{Q}(s) + \gamma_c \beta \tilde{H}(\overrightarrow{f}(s, a)) \tag{4}$$

3.2. Refining Directions for A2C with Partially Sequential Memory Replay

For a basic A2C algorithm, the actor is represented by the parameterized policy π. The critic is represented by the approximator $\overline{V}(s)$ parameterized by a neural network. The critic approximates the value function V_π,that is equal to the sum of future discounted rewards expected in a given state. A2C is an on-policy RL algorithm. As discussed in Section 2, unfortunately, human demonstrations and past experiences are generated by different policies. The propagation of discrepancy between the current policy and previous policies might lead to a bad performance as shown in figure 3. This is kind of like estimating the expected value of a random vector with one distribution, but samples are from different distributions.

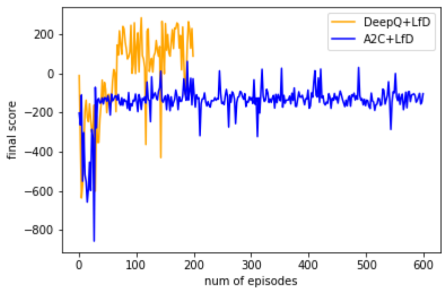

Figure 3. Comparisons of DQN/A2C + simple non-sequential memory replay.

In order to get unbiased updates for actor network and critic network, we need to refine network update directions to compensate the policy discrepancy in demonstrations/experiences and current status [17, 18].

3.2.1 Randomized-Truncated Estimators

To design unbiased estimators of actor and critic network updates, we have to guarantee that their variances are bounded and their bias asymptotically vanish. This is the only way for the algorithm to preserve the limit properties of convergence. Let $b > 1, \theta^a_{i+j}$ be the actor network vector applied to generate action a_{i+j}, θ^c be the critic network vector and K be drawn independently from Geom(ρ). We introduce the randomized-truncated estimators for the actor and critic network parameter update directions, which are of the same generic form [19, 18]:

$$\sum_{k=0}^{k} G(\theta^a, \theta^c)\alpha^k z_{i,k} \min\left\{ \prod_{j=0}^{k} \frac{\pi(a_{i+j}; s_{i+j}, \theta^a)}{a_{i+j}; s_{i+j}, \theta^a_{i+j})}, b \right\}. \tag{5}$$

[18] proves this update rule is asymptotically unbiased and of bounded variance.

3.2.2 Partially Sequential Replay

Sequential replay is required to obtain unbiased updates. But sequential algorithms do not exploit all the information contained in the data and are known to require more environment steps to obtain a satisfactory policy. Instead, we implement a partially sequential replay algorithm [18], which stores in a short-time memory and uses them in a certain process that updates synaptic weights. Thus, it can achieve higher efficiency.

3.2.3 Four Proposed Ways to Improve A2C

- Reconstruct data from the nearest N sequential replay with similar policies
- Importance sampling draws observations generated by from current policy to compensate the discrepancy
- Randomized-truncated estimators guarantee that their variances are bounded and their bias asymptotically vanish
- Refine direction of parameter updates for A2C networks to reduce bias

3.3. GAIL

3.3.1 Network Architecture

To achieve a fair comparison between the proposed method and our baseline, the GAIL is designed using relatively shallow networks which contains almost the same number of parameters as our baseline has. Also, no deep learning tricks such as skip connections or batch normalization are used to boost the performance. The discriminator is composed of three layers of fully-connected layers with 128 hidden nodes. ReLU is used as the activation function. The input is simply as a $(n+m)*1$ vector where n represents the number of possible actions and m represents the length of each state vector. Before feeding the state into discriminator, the algorithm randomly sample Gaussian noise and add it to the state vector. The output of discriminator is a scalar from 0 to 1 which is the reward A will be fed back to generator for learning. We also use three layers of fully-connected layers in the policy network. Relu is used as the activation functions.

3.3.2 Proximal Policy Optimization Algorithm

Instead of the Trust Region Policy Optimization algorithm [20] utilized by J. Ho et al. [7], we utilize a relative simple but effective algorithm, proximal policy optimization (PPO) [21]. The main design element of PPO is its loss function which is shown as following,

$$L_t = E[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (6)$$

Where S denotes the entropy bonus of cations and L_t^{VF} is a squared-error loss before predicted value and target value. The composition of term L_t^{CLIP} is shown as following,

$$L_t^{CLIP} = E[r_t(\theta)\hat{A}_t] \quad (7)$$

Where $r_t(\theta)$ is probability ratio of current policy and previous policy and A is denote as the advantage of Q value over average values. Clipped surrogate objective prevents too large update at once and penalty on $r_t(\theta)$ which is probability ratio of current policy and previous policy deviating from 1, which is the most important design element of the algorithm.

3.3.3 Three Proposed Improvements to GAIL

To implement our GAIL algorithm on the environment of OpenAI luner lander, there are still some challenge unsolved, including 1) Bias in human's demonstrations, 2) Rewards are only generated by discriminator which might not be correct reward to give since the discriminator is learning at the same time, 3)Randomly sampling dataset ignores the time information of states and actions, 4)The biggest reward will only been discovered when the game comes to the end which the algorithm might not be able to explore.

To alleviate the above problems, we propose three simple tricks, the efficiency of which will be shown in the next several sections.

- Introducing a good teacher - balancing the reward from D and reward from environment
- Learning the time sequence - utilizing 1 dim convolutional layers
- Showing successful instance to policy network for initial the parameters of the network

The first idea is inspired by Pix2pix GAN[22] proposed by P. Isola et al. The paper utilize L1-norm as part of the lost function to train the generator which avoid turning the image to blurry. Following this idea, we also add the real reward to the reward give by the discriminator and use a hyperparameter λ to balance the rewards. The second idea is to learn the time sequence of state and actions pairs to generate better actions. By combining 5 continuous space state together, we are able to train a network to have the ability to generate action based on 5 states instance. 1d conlutional layer is much easier to train and tune compared to recurrent neural network without losing much performance [23]. Finally, to get a better initialization of the parameters, we will first pre train the network a little bit. The figures in the following sections will show that the performance of the algorithm largely depend on itself instead of pretraining.

4. Experiments

4.1. Data Collection

To train our LfD algorithms, we recruited 13 participants, and collect 142 success runs with 39672 (state, action, reward) pairs as human demonstrations and 95916 TAMER feedbacks. The human demonstrations are collected while the participants playing the game, we record their keyboard inputs, which corresponds to different actions, the current state, and the environment reward. To collect TAMER feedbacks, we encoded different keys on the keyboard with positive/negative rewards, or positive-confident/positive-not-confident/negative-not-confident/negative-confident for confidence method, and collect the current state, current action, and the human feedbacks (along with confidence value if needed). For TAMER feedbacks, we collect both confidence and feedback in practice this time, and discard the confidence values when training with GMM and Bayesian methods.

4.2. Baseline

To make a comprehensive comparison, we utilizes a deep Q-learning with human demonstrations warm started with memory replay as our baseline. The demonstrations are loaded in memory buffer, and the network is updated 390 steps with memory replay before it begins to interact and train with the environment.

4.3. TAMER

For all three methods of TAMER improvements, the TAMER feedbacks (as well as confidence values for confidence method) are loaded in memory buffer and the network is updated 390 steps with memory replay. For GMM method, the GMM is trained once with all the feedbacks and stay fixed. Then, the agent begins to interact and train with the environment. During the training, TAMER network is also updated using memory replay for each step although there won't be any new feedback coming in.

4.4. Refining Directions for A2C

The pseudo code is presented in Algorithm 1.

Algorithm 1: Refining Directions for A2C with Partially Se-quential Memory Replay

```
Initialize actor and critic networks, t = 1;
Execute an action under current policy or draw an action
  from human demonstrations;
Append observations in the database;
Only keep N most recent experiences in the same episode
  remain in the dataset;
for $m$ in $M$(replay intensity) do
    for $i$ in $N$ do
        Refining Directions for $\theta^a, \theta^c$ with importance
          sampling by eqn (5) ;
        Update $\theta^a, \theta^c$ with refined directions;
    end
    Set t = t+1 and repeat from step 2 until convergence;
end
```

4.5. Generative Adversarial Imitation Learning

The pseudo code is presented in Algorithm 2.

The training process takes CPU i7 to run 30 minutes for 10000 episodes.The gradient descent optimizer is Adam optimizer with learning rate of 0.001. In each iteration, we train the discriminator once and the generator twice to keep the training stable.

Algorithm 2: Improved Generative Adversarial Imitation Learning

```
Input: Expert trajectories $\tau_E$ $\pi_E$, initial policy and
  discriminator parameters $\theta_0$, $w_0$;
Pretrain the policy network until one win occurs;
for $i$ = 0, 1, 2,..., do do
    Sample trajectories;
    Update the discriminator parameters from $w_i$ to $w_{i+1}$
      with the gradient;
          $E[\nabla log D_w(s,a)] + E[\nabla log_w(1 - D_w(s, G(s)))]$
      Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the PPO
      update rule combine the loss rendered from the
      environment ;
end
```

5. Results and Discussion

5.1. Baseline

The results for baseline method is shown in Figure 4 and Table 2. It's clear that the human demonstrations have greatly accelerated the training process. The help of human demonstrations are coming in two ways. First, the agent starts interacting with the environment with a better initial policy, which is trained with human demonstrations using memory replay. Judging from the performance of the agent in early period of the training, this advantage is not very significant. Another way the agent takes advantage of human demonstrations is during the interaction, when doing memory replay, the replayed part of memory is drawn randomly from the memory buffer, which contains large amount of human demonstrations. Therefore, the network will be updated with the demonstration along with environment rewards, which may help shape the policy faster because the human demonstrations contain a large number of different states, which gives the agent a great deal of exploration without trying to actually explore the environment.

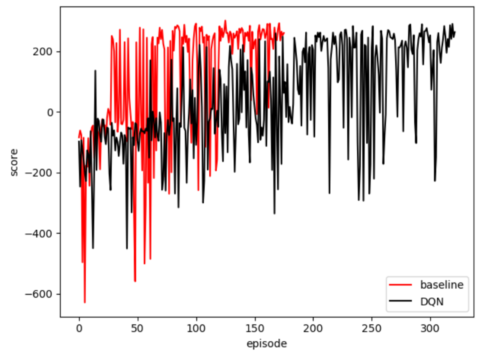

Figure 4. Comparison between baseline and DQN

Algorithms	Baseline	DQN
Episode	176	322

Table 2. Comparison between baseline and DQN

5.2. TAMER

The results of different methods are shown in Figure 5 and Table 3. The GMM method achieves the best result among the three, demonstrating a significant improvement upon plain TAMER and achieving a result close to the baseline, while the results for confidence and Bayesian methods are not very good.

We think the reason why confidence network fails to show large improvement is because human teachers don't know how confident they are about the confidence they give just as they don't know how confident they are about the feedbacks they give. Also, since human teachers are not perfect experts, the action they prefer may not be the optimal action in a specific states. Therefore, they may be confident in a wrong action, and give a negative feedback with strong confidence to the correct one.

As for the Bayesian methods, there is an underlying assumption for this method that the distribution of the rewards for each actions is Gaussian Distribution. However, this assumption may only be true when we have a large enough dataset. Although the collected TAMER feedbacks seem to large, this may not be enough because the state space is even larger. Therefore, when training the network, there may not be enough data for every groups of similar states.

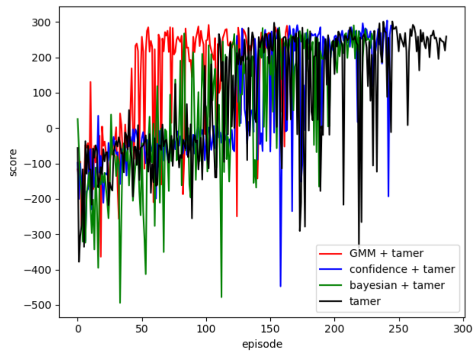

Figure 5. Comparison between GMM, confidence, bayesian, and plain TAMER

Algorithms	GMM	confidence	Bayesian	TAMER
Episode	173	245	231	288

Table 3. Comparison between GMM, confidence, bayesian, and plain TAMER

To further test the proposed method, we conducted further experiments on the GMM method, where the method is combined with human demonstrations as warm start. The use of human demonstrations is the same as that in the baseline. And the results are shown in Figure 6 and Table 4. In this further experiment, the GMM method combined with human demonstrations as warm start greatly outperforms the baseline method. Also, as can be seen in the figure, warm started GMM successfully lands the agent in early stage of training, which is a sign of good initial policy. This is because in the early stage of training, a good initial policy with predicted human reward will encourage the agent to take actions similar to human actions, which generally aims to safely land the agent with zero velocity in the landing pad. However, this combined method may also fail if the GMM + TAMER fails to estimate a good human reward function, which may be the case when there are too many conflicting or wrong feedbacks. And as the number and diversity of the human teachers grow, the feedbacks may also become more "diverge", maybe containing more conflicting feedbacks because of different personal "preferences" of different teachers.

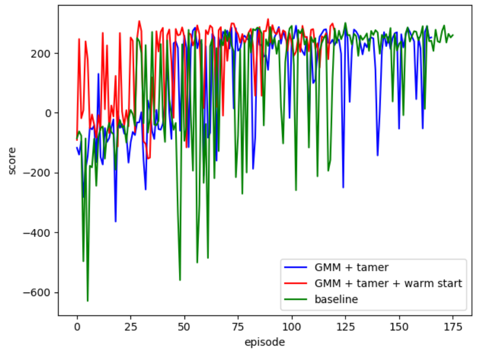

Figure 6. Comparison between GMM, GMM with warm start, and baseline

Algorithms	GMM	GMM+demo	Baseline
Episode	173	121	176

Table 4. Comparison between GMM, GMM with warm start, and baseline

5.3. Refining Directions for A2C

To illustrate the efficiency of different RL algorithms that can take advantages of human demonstrations through memory replay, and analyze underline characteristics of memory replay, we compare the DQN, vanilla A2C, and A2C with refined directions. Results are shown in figure 7. DQN with non-sequential memory replay achieves the best result, vanilla A2C with non-sequential memory replay performs the worst, and A2C with refined directions and partially sequential memory replay outperforms the vanilla A2C, but is not better than DQN.

First, the best performance from DQN shows that off-policy RL algorithms can make the best advantages of human demonstrations through memory replay. Because human demonstrations from off-policy RL generated from the same policy in expectation. We can efficiently exploit them by non-sequential replay and recursively use high-quality human data. Thus off-policy RL is extremely suitable to exploit human demonstrations through memory replay.

Second, the worst performance from vanilla A2C demonstrates the limitation of on-policy RL algorithms to exploit human demonstration data. Since the human demonstration data generated by different policies, and it introduces

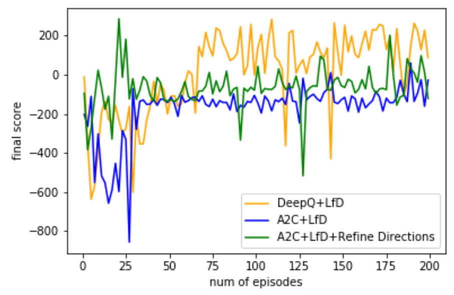

Figure 7. Comparisons of DQN, A2C, and A2C with refined directions. DQN and vanilla A2C uses non-sequential memory replay, A2C with refined directions does partially sequential memory replay.

discrepancy between the current policy and previous policies. In return, the replay cannot help the learning.

Third, the improvements of A2C with refined directions come from the four proposed techniques, that reduce bias in the memory replay updates and enable the algorithm to exploit the human data generated by different policies. Proposed randomized-truncated estimators guarantee that their variances are bounded and their bias asymptotically vanish. This crucial technique makes it possible to replay past experiences. Moreover, partially sequential replay utilizes the most recent memory, and approximates that they are generated by the same policy. Thus, the improvements boost the efficiency of the on-policy RL algorithm. Moreover, it totally makes sense it works worse than DQN. Because the on-policy RL algorithm can only take the partially sequential experiences within the same episode, which is less efficient for off-policy RL algorithm that can totally sequential experiences to replay.

5.4. Generative Adversarial Imitation Learning

To check the effectiveness and efficiency of GAIL, we first training the original GAIL without proposed improvements. The training results in shown in figure 8. From the figure

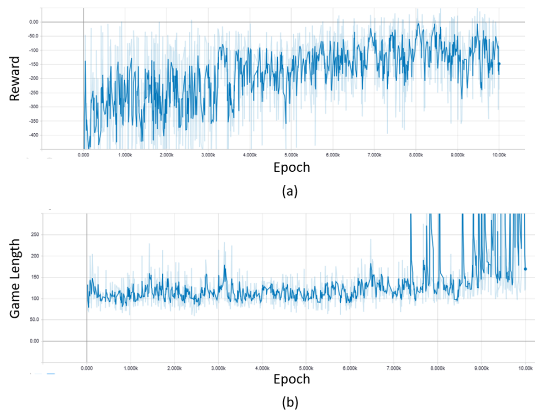

Figure 8. The training result of original GAIL of 10000 epoch. (a) Reward of each epoch. (b) The length of game of each epoch.

(a) we can see that almost all the sum of rewards of each epoch is negative which mean the agent has never win in 10000 training epoch. From the figure (b), we see that the length of each round keep almost the same for the first half of the training. However, in the late one third of the training the length increases dramatically. Many of them reach out of the threshold of the game and the game is ended automatically. Thus, we can say that in the end the training, the agent learns a local optimal about how to get more reward, which is to stay alive in the game and doing back and forth to get small rewards. However, in the all 10000 epochs, there is no even one time when the agent lands on the ground safely which shows some backwards of the algorithm including those mentioned in section 3.

Then we do experiments on the improved GAIL to see if the new version will alleviate the problems. The results are shown in figure 9. From the figure (a), we can observe that

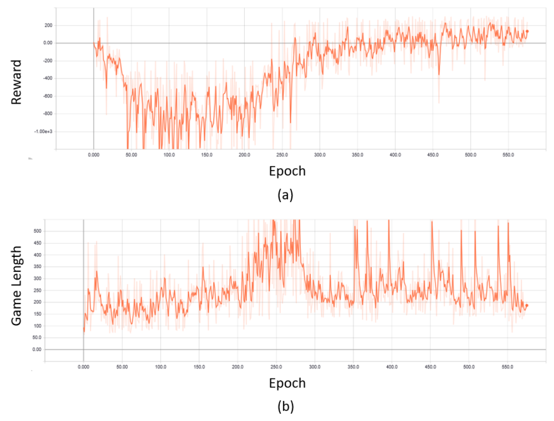

Figure 9. The training result of improved GAIL until it achieves the goal which takes 575 epoch. (a) Reward of each epoch. (b) The length of game of each epoch.

the reward curve goes down and go up and achieves the goal with 575 epoch. The process of going down and back indicates the learning process of the algorithm. From the figure (b), we can observe a interesting learning experience of the algorithm that it at first learn to stay in the game to get more rewards. Then, it find that landing safely will give higher reward and turn to learn how to land saving quickly after that. The results shows that with simple revision on the original GAIL, we achieves pretty good results. However, there is still on question is not answered. Since we are using pretrain as a tool for weights initialization, is it when the algorithm learn how to success or it is in the former training process? To answer this question, we also show figure 10. From the figure, we can see that after

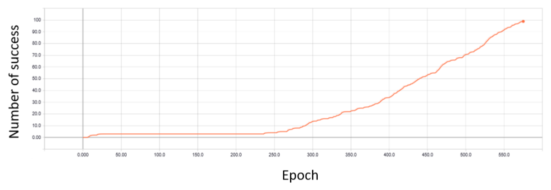

Figure 10. The number of success along with the training epoch

pretraining, the number of successful trail increases very slow. Not until the later half, the number of successful trail starts increasing. This shows that pretraining is just giving the network a good parameter initialization.

Algorithms	GAIL	RR	1dCNN	PT	GAIL+all
Episode	N/A	N/A	N/A	3229	575

Table 5. Comparison between different algorithms related to GAIL in this project. RR is GAIL with real reward from environment. PT is GAIL with pretraining. CNN is GAIL built using 1d CNN. All is GAIL using all three tricks. N/A means the results might be larger than 10000 epoch.

The final evaluation results are shown in Table 5. The evaluation results of improved GAIL is not as strong as other methods mentioned in the project. On one hand, we are using simple operators in GAIL to avoid have a network with a bulky size. GAIL with very large networks will be hard to train, resource-costly and unfair to the baseline model. One the other hand, the instability training of GAN might be the main reason why GAIL is not a fast learner, since the main goal is coded ambiguous to the generator.

6. Conclusion and Future Works

In this work, we implemented and improved three different LfD algorithms, and conducted experiments to demonstrate the improvement and compare the results to our baseline method.

6.1. TAMER

All three improvement methods have shown their effectiveness, especially the GMM method. Both the confidence method and the Bayesian method have shown improvement compared to plain deep TAMER, but they are not competitive compared to the baseline method, whereas the GMM method shows a similar performance as the baseline and outperforms it when trained with human demonstrations as warm start.

However, there are still some problems and possible future works with the methods. First, the collected human feedbacks are directly loaded in the memory buffer and trained using memory replay, which may not be very reasonable given that one of the advantages of TAMER framework is that it distributes the influence of human feedback among all the past experiences in a certain time window. There may be better ways of handling these collected feedbacks, such as collecting both human feedbacks and past experience in a certain time window for each feedback, then simulate a TAMER feedback when training. Also, the influence of individual difference in action preferences may introduce further conflicts. And a further problem with the GMM method now requires parameter tuning for cluster number. It may be better to make it as a learnable parameter. Another possible way is to use some heuristic to measure the confidence of the feedbacks. This method has the potential to perform well with data collected from large number of participants because it's robust to individual preference difference.

6.2. Refining Directions for A2C

We successfully combine the technique of memory replay with A2C to exploit the human data with efficient partially sequential update, even thought they are initially generated under different policies. These algorithms deserve serious attention since A2C represent one of the most successful approaches to apply reinforcement learning to realistic control tasks. To make this combination correct for the on-policy algorithm, all four proposed techniques ensure that the discrepancy between current policy and the policy used at the time when the replayed experience was generated is compensate.

To further improve the algorithm, there are two possible directions. One way is to design better sampling techniques for the direction estimators that can converge faster. The other way is to add a step-size estimation. Those designs might be case dependent.

6.3. Improved Generative Adversarial Imitation Learning

GAIL as a generative model. It is supposed to learn the underlying data distribution of the ground truth data. Here, we apply it to learn the policy of playing an OpenAI gym game which is relatively challenging than the benchmarks mentioned in the original paper. Besides completing the game, we also test its ability to learn fast. The results show that our improved GAIL can achieve a relatively good result with very small size of deep learning architecture. However, compared to other algorithms, GAIL is not a fast learner. This is very likely because of the instability of the training process of GAN. It requires a lot of engineering and efforts to fine-tune the algorithm to get a nice performance.

In the future, one interesting topic to try is using meta learning or transfer learning. Actually, for warm start, it is a very common tool in transfer learning. To train a fast learner, one solution will be to have some pre knowledge. Then you can learn any related topic fast enough once you master the method to transfer knowledge to new scenario.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. In *Advances in Neural Information Processing Systems*, pages 8011–8023, 2018.

[3] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 478–485, 2017.

[5] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[8] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.

[9] W Bradley Knox and Peter Stone. Combining manual feedback with subsequent mdp reward signals for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 5–12. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[10] W Bradley Knox and Peter Stone. Reinforcement learning from simultaneous human and mdp reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 475–482. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[11] Hirotaka Hachiya, Jan Peters, and Masashi Sugiyama. Reward-weighted regression with sample reuse for direct policy search in reinforcement learning. *Neural Computation*, 23(11):2798–2832, 2011. PMID: 21851281.

[12] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[13] Nir Baram, Oron Anschel, Itai Caspi, and Shie Mannor. End-to-end differentiable adversarial imitation learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 390–399, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[14] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[15] Ilya Kostrikov, Kumar Krishna Agrawal, Sergey Levine, and Jonathan Tompson. Addressing sample inefficiency and reward bias in inverse reinforcement learning. *CoRR*, abs/1809.02925, 2018.

[16] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *CoRR*, abs/1807.06158, 2018.

[17] Paweł Wawrzyński and Ajay Kumar Tanwani. Autonomous reinforcement learning with experience replay. *Neural Networks*, 41:156 – 167, 2013. Special Issue on Autonomous Learning.

[18] Paweł Wawrzyński. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 22(10):1484 – 1497, 2009.

[19] V.R. Konda and J.N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003. cited By 328.

[20] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[22] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[23] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018.