Take full advantage of demonstration data in deep reinforcement learning

Yong-Xu Zhang

1 College of Computer science, Sichuan University, Chengdu, Sichuan, 610065, China
* Corresponding author’s e-mail: zhangyongxu@stu.scu.edu.cn

Abstract. Deep reinforcement learning (DRL) algorithms have achieve a great breakthrough in many tasks, even though, they still suffer from the problems, such as random exploration and sparse reward. Recently, some reinforcement learning from demonstration (RLfD) methods have shown to be promising in overcoming these problems. However, they usually require considerable demonstrations and the demonstration data may not be optimal which means the absent of exploitation. To deal with it, we propose a novel algorithm in which we use a behaver cloning network BC to learning the demonstration data, and then use it to guide the reinforcement learning agent, when the agent learn and explore a better trajectory, the BC then learning the trajectory to make itself better, which will lead the BC network and the agent become better at the same time.

1. Introduction
Deep reinforcement learning (RL) has made it possible to build end- to-end learning agents without having to human labelled data, as it performs well in various challenging domains such as MOBA games in the recent years. These feats make deep RL a great candidate to be employed in complex real-world sequential applications. However, while deep RL algorithms enable the use of large models, the use of large datasets for real-world RL is conceptually challenging. Most RL algorithms collect new data online every time a new policy is learned, which limits the size and diversity of the datasets for RL.

An off-policy RL agent needs to explore the environment for gathering training samples, which makes the learning process potentially costly or even unable to converge for many real-world applications. For this reason, many of the recent investigations in the area aim at reducing the amount of required samples during learning.

Deep Q-learning from demonstration[1], also known as DQfD, leverage the expert demonstrations to pre-train the Deep Q-learning agent, which is discussed in [9] and [10] as well, showing the benefit of demonstration, then use the pre-trained model to interact with the environment to collect self-generated data. This paradigm has had successes in many challenging robotics applications. However it still faces some problems as well, such as the demonstration data is hard to collect and it may not be optimal[6], and the demonstration data does not reuse the second phrase any more[7].

In order to solve the problem, in this paper, we propose a novel paradigm to reuse the demonstration data, and take advantage of the agent exploration, at the same time. We use a behaviour cloning network, which can view as a map from state to action to record the demonstration data, and use this network to giving the agent some advice. We evaluate the demonstration data and the behavior cloning network at the first time, when the agent can be have a better policy which surpass the behavior cloning network, the behavior network learns this specific trajectory. The RL agent and the BC network learn from each other and make progress and the same time, make the final result outperform.
2. Background

2.1. Markov Decision Process
An MDP[3] is defined by a tuple \((S, A, R, T, γ)\), tuple which consists of a set of states \(S\), a set of actions \(A\), a reward function \(R(s, a)\), a transition function \(T(s, a, s') = P(s'| s, a)\), and a discount factor \(γ\). In each state \(s \in S\), the agent takes an action \(a \in A\). After taking this action, the agent receives a reward \(r\) from \(R(s, a)\) and reaches a new state \(s'\), this new state \(s'\) is determined from the probability distribution \(P(s'| s, a)\). A policy \(π\) specifies for each state which action the agent will take. The goal of the agent is to find the policy \(π\) mapping states to actions that maximizes the expected discounted total reward. The value \(Q(π(s, a))\) of a given state-action pair \((s, a)\) is an estimate of the expected future reward that can be obtained from \((s, a)\) when following policy \(π\). The optimal value function \(Q^*(s, a)\) provides maximal values in all states and is determined by solving the Bellman equation:

\[
Q^*(s, a) = E\left[R(s, a) + γ \sum_{s'} P(s'| s, a) \max_{a'} Q^*(s, a')\right]
\]

The optimal policy \(π\) is:

\[
π(s) = \arg\max_{a \in A} Q^*(s, a)
\]

2.2. Deep Q Learning
The DQN[2] use a deep neural network, or Q net to approximates the \(Q(s, a)\). Besides it, DQN use a policy network \(θ\) to generate action in any state, and a target network \(θ'\), which is a copy of the Q function (or the previous policy network) that is held constant to serve as a stable target for learning for some fixed number of timesteps. The policy network is trained using the loss:

\[
loss = E\left[(r + γ\max_{a'}Q^θ'(s', a') - Q^θ(s, a))^2\right]
\]

After every timestep \(t\), target network will be synchronized with the target network. To reuse the self-generated data, DQN proposal a replay memory buffer, during learning phrase, the agent uses the samples from this memory. It can also prevent action values from oscillating or diverging, which can be view as an essential mechanism in DQN

3. PROPOSED APPROACH
Consider a person begin to learn something new, such as play a new game, a new sport or learn a new skill, usually there is a guider or a teacher who teach the person to learn. At the very beginning, he will look at how the experts do, and then try it himself, at the same time, he will continue observe the experts until he surpasses the experts. Once the experts notice someone else is better, the experts will learn the better experience to improve themselves. Identical, we use the scenario at our algorithm, consider a behavior cloning is a expert at the first phrase, when a RL agent keep learning, it will collect some extremely useful experience, but it can not learn it because it may not be sample during learning stage. In the next time, the agent faces the same state, it will choose some action else, which is not what we want, at this time, our BC will give another action advice which is learning from the old trajectory.

When our agent become better and better, it can perform very well by itself, it will not rely on the BC network anymore, so we decrease the probability to use the BC’s guide advice. Whereas at the very beginning, the agent will prefer to take the BC’s advice. However, a BC is not a perfect guider, we decrease the taking guide probability when a guide advice don’t perform well in the trajectory, to make the agent more independent, and explore by itself. Which means when facing a bad guider, the agent is inclined to explore by itself, or learn by itself rather than learning from someone else.

To separate the experience which is guide by the BC network, we use another replay buffer which is called \(R^{uid}\) to store the guiled data, we think the guiled data is prioritized, so when using the guiled data, we use a different to calculate the loss. The \(J_E\) loss:

\[
J_E = \max_a [Q(s, a) + l(a_E, a)] - Q(s, a_E)
\]
where $a_E$ is the action the expert demonstrator took in state $s$ and $l(a_E, a)$ is a margin function that is 0 when $a = a_E$ and positive otherwise.[1]

Algorithm 1 BC DRL Algorithm

1. **Input:** $\mathcal{D}_{\text{demonstration}}$: Demonstration data, $BC$: behavior cloning network, $\theta$: weights for initial behavior network (random), $\theta^t$: weights for target network (random), $r$: frequency at which to update target net, $k$: number of behavior cloning network training gradient updates, $m$: number of initial behavior cloning network training gradient updates $s$: number to evaluate the behavior cloning network $R^{\text{replay}}$, replay buffer, $R^{\text{build}}$: guild replay buffer, $\lambda$: trajectory of a episode, $t$: number of times to sync the target network $p$ to take action generated by $BC$ network
2. for step $t \in 1,2,3,..,m$ do
3. train BC network with data from $\mathcal{D}_{\text{demonstration}}$
4. end for
5. using the $BC$ to interact with environment $i$ times, calculate the average reward $R_{BC}$
6. for episode $e \in 1,2,3,..$ do
7. set $R_e = 0$ and $\lambda = \text{null}$
8. for step $t \in 1,2,3,..$ do
9. Sample action $a_e$ from behavior policy $\pi$
10. Sample action $a_{BC}$ from behavior policy $BC$
11. Take action from $a_{BC}$ with probability $p$ if $a_e \neq a_{BC}$
12. Store $(s, a_e, r, s')$ into $R^{\text{replay}}$ or $(s, a_{BC}, r, s')$ into $R^{\text{build}}$, and store $(s, a)$ in lambda
13. Calculate the total reward $R_e += r$
14. Sample a mini-batch of $n$ transitions from $R^{\text{replay}}$ and calculate loss $J$ or from $R^{\text{build}}$ and calculate loss $J(E)$ using target network
15. Perform a gradient descent step to update
16. if $t \mod r = 0$ then $\theta' \leftarrow \theta$
17. end if
18. end for
19. if $R_e \geq R_{BC}$ and take action $a_{BC}$ then update the $BC$ with data $\lambda$ for $k$ times and decrease the probability $p$
20. end if
21. if $R_e \geq R_{BC}$ and never take action $a_{BC}$ then update the $BC$ with data $\lambda$ for $k$ times and decrease the probability $p$
22. end if
23. if $R_e < R_{BC}$ and take action $a_{BC}$ then decrease the probability $p$
24. end if
25. if $R_e < R_{BC}$ and take action $a_{BC}$ then increase the probability $p$
26. end if

Figure 1 pseudo-code about this algorithm.

4. EXPERIMENTAL

4.1. Experiments Configuration

In this paper, we are using some simple environment to interact. The demonstration data was collected by manual marking. In the game MountainCar-v0, A car is on a one-dimensional track, positioned between two "mountains", the agent got a -1 reward each timestep until it reach the flag position, when reach the goal place, it will get a 0 reward, and the max timestep for the environment is 200 by default. For the CartPole-v0 game, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track the agent got a +1 reward each timestep and the max timestep for the environment is 200 as well. The actions for both environments are same: left and right, or zero and one. Observation
space dimension is a vector which describe the physical state of the current state such as velocity and height (for the MountainCar-v0) or angular velocity and angle (for the CartPole-v0).

![Figure 2 Classical control tasks environments in OpenAI gym used in this paper. MountainCar-v0(left), and CartPole-v0(right)](image)

For evaluating the learning performance in this domain, we use DQN as the base learning algorithm and at very begin, we play the game for several to collect the demonstration data, subject to the experience of playing the game, we don’t get perfect demonstration data to feed the agent, however, our work do reduce the random exploration in some extent.

Then we use the demonstration data to first train the BC network, and use it to interact with the environment. With the simple environment, the BC network can sometimes be very strong to achieve a score of 200 in CartPole-v0, but sometimes it may get a 0 score because it can’t handle with some known state. So we use a mean of the reward to use, this is a baseline or $R_{BC}$ in above pseudo-code[Figure 1].

We train the agent for 1000 game episode in total, where the agents are evaluated (exploration and updates turned off) for 10 episodes at every 10 learning episodes. With each episode, we take 200 action timestep, in other word, each game episode will take actions for 200 times. Each timestep 4, we will sample some data from $R_{replay}$ or $R_{guid}$. We set the $l(a_E, a)$ to be 0.8 for both environment, and set $\epsilon$ to be 0.2 wich means it will explore the environment with probability 0.2. We reduce the $\epsilon$ every time it sample from the $R$ to updates the parameters in the Q network. Once a trajectory get a total reward that is greater than the $R_{BC}$, we use this specific trajectory to train our BC network for k times, here we set k is equal to 50. Each time we want to reducing the probability of taking action from the BC network, it reduce 0.01, and we set the initial k to be 1, which means the agent learn from the BC totally, and then when we reduce the probability, the agent then can take $\epsilon-greedy$ algorithm to take actions to Exploration and Exploitation.

4.2. Experiments Result
We use DQN as our baseline algorithm, DQfD is involved, so we use the DQfD as well, but restrict to the struct of our demonstration data, we can’t perform all the loss function in DQfD, but somehow we still use DQfD’s training method.

Here are some tables and figures about those three algorithm.

For the first environment, MountainCar-v0, the result are as follows.
Figure 3 Performance of our method versus DQN and DQfD in environment of game MotainCar-v0

Table 1 Comparative result for the algorithm perform in MountainCar-v0.

|                  | first surpassing human episode | Mean reward | Max reward |
|------------------|--------------------------------|-------------|------------|
| DQN              | 121                            | -137        | -84        |
| DQfD             | 30                             | -135        | -86        |
| ours             | 18                             | -125        | -86        |

For the another environment, CartPole-v0, the result are as follows.

Figure 4 Performance of our method versus DQN and DQfD in environment of game CartPole-v0

Table 2 Comparative result for the algorithm perform in CartPole-v0.

|                  | first surpassing human episode | Mean reward | Max reward |
|------------------|--------------------------------|-------------|------------|
| DQN              | 121                            | 155         | 200        |
| DQfD             | 1                              | 170         | 200        |
| ours             | 1                              | 173         | 200        |

We notice that our algorithm and DQfD can learn quickly with demonstration data. And notice ours can over perform compare to the others.

4.3. Experiments Analysis
Since the above figures and tables show that our method work, but it still exists some problems, which means our methods may not be flawless, but in these experiments, it shows its ability to accelerate the learning process, especially compare to DQfD, it doesn’t need a pre-training process, just like a normal DQN. In the CartPole-v0 environment, the total reward has experienced oscillation around episode 150, which is the guider’s responsibility, when the BC network choose a action at a unknown state, it may choose absolutely wrong actions, however, the agent and the BC network can’t get any perception any
all, fortunately, after reducing the probability, the agent recover to a DQN level agent, and what BC
guide can still vie as a kind of exploration.

In the mountain car environment, benefit from the BC network’s guide, the agent doesn’t use much
time to take random actions, and quickly learn, and with the BC, the agent can be easily adaptive in just
a few times of gameplay in the real-world environment.

5. CONCLUSIONS
In this paper, we developed an approach to guide a reinforcement learning agent to efficiently interact
with the environment, we use the ideology of [5], using a manual demonstration data. Our study can be
viewed as a combination of take advice[8] and take demonstration. But we focus on how to use the
demonstration, rather than when to give the advice.

Since previous work have found that how to use the demonstration data can be a big challenge, we
use a BC network, because behavior cloning requires much less computation than RL, it only learn a
map from state to action, which can learn a better policy very quickly ignoring the sparse reward, at the
same time, the RL agent, using off-policy learning strategy, first learning from the imperfect BC network,
when the agent is still aware about the environment, it can output action by itself, to collect better
trajectory experience to learn. Meanwhile, the BC network taking the advantage of less computation and
easy to attain convergence, learn those better trajectory, make the BC to be a better guider to overcome
the sample efficiency.

References
[1] Hester, Todd, et al. "Deep q-learning from demonstrations." Proceedings of the AAAI
Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.
[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature
518.7540 (2015): 529-533.
[3] Sutton, R. S., and Barto, A. G. 1998. Introduction to reinforcement learning. MIT Press.
[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W.
(2016). Openai gym. arXiv preprint arXiv:1606.01540.
[5] Argall, Brenna D., et al. "A survey of robot learning from demonstration." Robotics and
autonomous systems 57.5 (2009): 469-483.
[6] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. 2018. Reinforcement
Learning from Imperfect Demonstrations. In 6th International Conference on Learning
Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop
Track Proceedings.
[7] Nair, A., Dalal, M., Gupta, A., & Levine, S. (2020). Accelerating online reinforcement learning
with offline datasets. arXiv preprint arXiv:2006.09359.
[8] Silva, F. L. D.; Glatt, R.; and Costa, A. H. R. 2017. Simultaneously Learning and Advising in
Multiagent Reinforcement Learning. In Proceedings of the 16th International Conference on
Autonomous Agents and Multiagent Systems (AAMAS), 1100–1108.
[9] Xiao-Yong Wei, Zhen-Qun Yang, "Coaching the Exploration and Exploitation in Active Learning
for Interactive Video Retrieval", IEEE Transactions on Image Processing, vol. 22, no. 3, pp.
955-968, 2013
[10] Xiao-Yong Wei, Zhen-Qun Yang, "Coached Active Learning for Interactive Video Search", ACM
Multimedia 2011 (MM’11), Scottsdale, Arizona, USA, November 28-December 1, 2011