IMPROVING LEARNING FROM DEMONSTRATIONS BY LEARNING FROM EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

How to make imitation learning more general when demonstrations are relative limited has been a persistent problem in reinforcement learning (RL). Poor demonstrations leads to narrow and biased date distribution, non-Markovian human expert demonstration makes it difficult for the agent to learn, and over-reliance on sub-optimal trajectories can make it hard for the agent to improve its performance. To solve these problems we propose a new algorithm named TD3fG (TD3 learning from generator) that can smoothly transition from learning from experts to learning from experience. This innovation can help agents extract a priori knowledge from demonstrations while reducing the detrimental effects of the poor Markovian properties of the demonstrations. Our algorithm achieve good performance in mujoco environment with limited and sub-optimal demonstrations.

1 INTRODUCTION

Reinforcement learning has been highly successful in many areas, meeting or exceeding human performance, in video game environments (Mnih et al., 2015), robot manipulators (Andrychowicz et al., 2018), and various open source simulation scenarios (Lillicrap et al., 2015a). The problem with reinforcement learning, however, is that it requires frequent interaction with the environment to collect data, which is difficult to do in most application scenarios. These mentioned are limited significantly by the need to explore potential policies through trial and error, which produces initial performance significantly worse than human expert. It make RL sample inefficient and slow to converge.

One way of overcoming these limitations is to utilize human expert demonstrations to provide agent with priori knowledge and initialize it to a higher level of performance compared with randomly initialized networks. This is often termed Learning from Demonstrations (LfD) (Argall et al., 2009), which is a subset of imitation learning that seeks to train the policy to imitate the behavior of human expert. Behavior cloning (BC) is of the LfD methods, which collect demonstrations for supervised learning and produce an agent that can generate eligible similar behavior in short training time with limited data(Goecks et al., 2019). BC attempts to approximate a map between state and action based on the state-action pairs form expert demonstrations and generate a policy that can mimic this behavior. Related research has shown that in areas such as unmanned vehicles, robot control, behavior cloning on large scale off-policy demonstration datasets can vastly improve over the state of the art in terms of generalization performance (Codevilla et al., 2019).

Though BC techniques do allow for the relatively rapid learning of behaviors, it can suffer a domain shift between the off-line training experience and the on-line behavior (Ross et al., 2011). Especially when the demonstrations are limited and not optimal the performance of BC dramatically decreased even generate negative impact in our experiments with subset of D4RL datasets. BC faces many challenges as an effective method to improve the efficiency of RL training, such as the fact that human-generated data is not strictly markov in nature, which often poses some challenges to the algorithm as well. Besides in real-world tasks, it is often difficult to obtain optimal trajectories, so we hope that RL can get a final performance boost from sub-optimal data, but over-reliance on sub-optimal demonstrations can have bad results (Wu et al., 2019).

And the distribution of data may be narrow or biased such as those from hand-crafted deterministic policies or arise in human demonstrations may cause divergence both empirically (Fujimoto et al., 2019) and and theoretically (Dulac-Arnold et al., 2019) (Fu et al., 2020).
In this work we propose a TD3 learning from generator (TD3fG) framework, which aims reducing the adverse effects of demonstrations sub-optimal and non-Markovian properties. Unlike previous approaches to combine BC with RL which use demonstrations to do supervised learning and directly initialize the agent's policy network, our method utilize demonstrations to train a small network as reference action generator and import the reference action into exploration noise and loss function part of TD3. This generator can generate undesirable but informative actions in unfamiliar states. Our method utilize the reference action with linear decreasing weights to realize a smooth transition from imitation learning to reinforcement learning. Our method increases the training speed and final performance in some mujoco tasks compared with original TD3, BC and some other methods that utilize demonstrations like DDPGfD. To summarize, the main contribution of this work.
• Propose a framework of TD3fG in section IV.
• In section VI we took 100 sub-optimal demonstrations from the d4rl dataset and ran a series of comparisons experiments with BC, DDPGfD, etc.
• In section VII we perform ablation experiments to investigate the contributions of the individual components in our method.

2 BACKGROUND

A. DDPG and TD3
Our method is developed based on Twin Delayed Deep Deterministic policy gradient algorithm (TD3). TD3 is a deterministic, off-policy and model free reinforcement learning algorithm which is suitable for continuous action control (Fujimoto et al., 2018). TD3 seen as a modified version of DDPG.
DDPG alternately interacts with environment to collect experience and update network's parameters (Lillicrap et al., 2015b). For every step, it selects action from $a_t = \pi(s_t|\theta_\pi) + \mathcal{N}_t$ according to current policy π and exploration noise N. After executing action a_t the agent observes reward r_t and new state s_t. It then stores the transition (s_t, a_t, r_t, s_{t+1}) in reply buffer R. During the training step, the agent samples a random mini-batch N transitions (s_t, a_t, r_t, s_{t+1}) from R Silver et al. (2016)Levine et al. (2016). According to Bellman equation, the target value of critic network output is:

$$y_i = r_i + \gamma Q(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'}) \tag{1}$$

Update critic parameters by minimizing MSE loss:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta_Q))^2 \tag{2}$$

Update the policy parameters using the sampled policy gradient:

$$\nabla_{\theta_Q} J \approx \frac{1}{N}\sum_i Q(s, a|\theta_Q)|_{s=s_i, a=a_i} \nabla_{\theta_\pi}\pi(s|\theta_\pi)|_{s=s_i} \tag{3}$$

Note that the actor and critic network interact with environment during the on-policy process while the target actor and critic network is used to update network parameters. After updating on-policy networks through the above method we need to soft update the target networks:

$$\theta_{Q'} = \tau\theta_Q + (1-\tau\theta_{Q'}),\ \theta_{\pi'} = \tau\theta_\pi + (1-\tau\theta_{\pi'}) \tag{4}$$

TD3 is a more advanced method. Its main improvement is the import twin critic networks to solve the

problem about Q value over estimation. It randomly initializes two critic networks and choose the minimum of the two outputs as the output Q value. Furthermore TD3 also adds noise to the target action. By smoothing the changes of the Q function along different actions, the policy is more difficult to be affected by the error of the Q function. After sample mini-batch N transitions (s_t, a_t, r_t, s_{t+1}) from R, modify action a according to

$$\tilde{a} \leftarrow \pi'(s|\theta_{\pi'}) + \mathcal{N}, y_i \leftarrow r_i + \gamma min_{i=1,2} Q_i(s_{i+1}, \tilde{a}) \tag{5}$$

Delayed update means in the TD3 algorithm, the update frequency of the policy (including the target policy network) is lower than the update frequency of the Q function. (It is recommended that the Q network update frequency be twice that of the policy network).

B. Behavior Cloning

Behavior cloning (BC) directly learns from expert demonstration. With states from demonstration as input and corresponding actions as target output, the agent can directly update policy network with supervised learning algorithm using MSE cost function

$$L_\pi = \frac{1}{N} \sum_i (a_{demo} - \pi(s_{demo}|\theta_p i)^2 \tag{6}$$

BC directly approximate the map from state to action without considering reward function (Nair et al., 2018). It is difficult to perform imitation learning from high-dimensional state-action space, and the normal behavioral cloning algorithm is prone to cause distributional shift (Ross et al., 2011). The agent does not know how to return to the expert track state which will lead to error accumulation (Zhang & Cho, 2016). Besides non-Markov behavior and multi-modal behavior also suppress BC's performance and play a negative role especially with poor demonstration (Kiran et al., 2021). This will be discussed in Section VI (EXPERIMENT).

3 METHOD

Our method combines TD3 with demonstrations in two ways to accelerate training and avoid restrictions of poor demonstrations. we use demonstrations to generate action noise and imitation loss.

A. Pre-trained reference action generator

First we use supervised learning and demonstrations to train a BC policy neural network. We assume that the the demonstrations is limited and sub-optimal. In experiments we prepare 100 expert demonstrations for every tasks and these demonstrations include relative good trajectories, sub-optimal trajectories and even some failed trajectories. The max, min and mean accumulated reward of demonstrations will be shown in experiment part. Since the amounts of data is small and demonstrations only cover part of action space and state space, we only use BC method to train the generator for 50000-100000 steps.

$$L_\pi = \frac{1}{N} \sum_i (a_{demo} - \pi(s_{demo}|\theta_p i)^2 \tag{7}$$

The generator can generate actions that are unreliable but with reference values. Experiment shows the generator performed unstably when directly used in tasks especially when states outside demonstrations.

B. Generate exploration noise

Second we use the pre-trained BC network to generate action noise. In this part we introduce the two parts of the noise in our method. TD3 trains a deterministic policy in off-policy method. Since the policy is deterministic, the agent may not try enough branch actions to find useful experiences at the beginning.

In order to make the TD3 policy explore better, we added a noise that consists of two parts, Ornstein Uhlenbeck process noise (OU noise) and reference action noise.
The BC policy networks can generate unstable but meaningful actions compared with completely random exploration actions. For example in Walker2D tasks, the random noise cause robot more likely to fall at early steps because the agent tends to explore action space equally in all directions. We use BC net to add a bias to the noise so that the agent can explore more purposefully and change the action distribution:

$$a_t \sim \mathcal{N}(a_t, \sigma^2),\ \hat{a_t} \sim \mathcal{N}(a_t + \alpha(t)\pi^{BC}(s_t), \sigma^2) \tag{8}$$

At early steps this trick can help agent collect more meaningful experience and accelerate training. But these noise will import negative effect when the agent has explored stable policy, So we need to decrease its weight along training. The total noise can be represented as:

$$n_t = \alpha(t)n_t^o + \beta(t)n_t^{BC},\ \alpha(t) = max(1 - \frac{t}{T_1}, 0),\ \beta(t) = max(1 - \frac{t}{T_2}, 0) \tag{9}$$

We use simple linear function $\alpha(t)$ and $\beta(t)$ as noise weights to balance exploration and exploitation. Generally T2 is bigger than T1 so the noise from generator converge to 0 faster. It's in line with the idea that the reference actions only work well in early steps.

C. Generate BC loss
Third, we propose a BC loss which is a MSE loss between output action and reference action from generator. This loss aims to provide a prior knowledge for agent exploration.

$$a_t^{BC} = \pi^{BC}(s_t|\theta^{BC}),\ L^{BC} = (a_t^{ref} - \pi(s_t|\theta^{\pi}))^2 \tag{10}$$

At early steps imitate the generator's actions can help agent shrink exploration scope. This apart is controlled by linear decreasing weight $\alpha(t)$. Meanwhile the weight of loss from critic increase along steps. This method allows the agent to gradually switch from imitation action to explore action space.

$$\begin{aligned} L &= \alpha(t)L^{BC}(a_t) - \beta(t)Q(s_t, a_t|\theta^Q)|_{a_t=\pi(s_t)} \\ \gamma(t) &= max(1 - \frac{t}{T_3}, 0),\ \delta(t) = min(1.2 - \gamma(t), 1) \end{aligned} \tag{11}$$

We import an extra hyperparameter T_3 which means after T_3 steps the agent is no longer affected by the generator. To simplify tuning we set $T_3 = T_2 = 0.5T_1$. Since the exploration noise generated using the OU process has a decreasing weight function, the range of exploration shrinks as the number of steps increases. So perhaps focusing on the RL loss part in advance will help speed up the training.The weight of the RL loss part can be modified as

$$\delta(t) = min(1 + \theta - \gamma(t), 1) \tag{12}$$

where we take the value of θ to be 2.
We also try to use Q-filter to decide whether to apply imitation loss. In the calculation of the loss function, the Q value of the reference action from generator is obtained through the critic first, then compared with the Q value of the output action of the policy network. If the reference action has bigger Q value we import the BC loss between policy output action and reference action. This method has no decreasing weights and associated hyperparameters, but then there is also no smooth transition on the loss function.

$$L^{BC} = 1_{Q(s_t, \pi^{BC}(s_t)) > Q(s_t, \pi(s_t))}(a_t^{ref} - \pi(s_t|\theta^{\pi}))^2,\ L = L^{BC} + L_Q \tag{13}$$

4 EXPERIMENT SETUP

A. Environments
We do several Mujoco simulations to evaluate our methods including Ant-v2, Walker2d-V2, HalfCheetah-v2

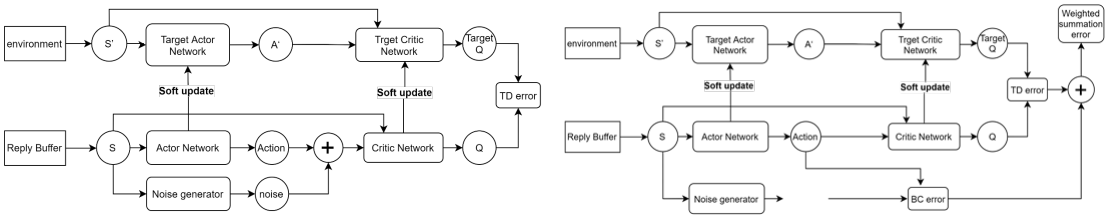

Figure 1: Network structure

and Hopper-v2. The task of Walker2d and HalfCheetah is to make a two-dimensional bipedal robot walk forward as fast as possible, but the robots model and reward function are different (Erez et al., 2011). Ant-v2's task is to Make a four-legged creature walk forward as fast as possible (Schulman et al., 2015). In all experiment we control biped or quadruped robots with different number of joints to walk as far as possible within specific steps. The reward function considers 4 aspects: forward reward (FR) which depends on how far the robot walks and health reward (HR) which depends on if the robot falls down after this step. Besides, the agent also receives penalty (negative reward) from contact and control cost (TC and CC).

$$r(s_t, a_t) = FW + HR - CC - TC \tag{14}$$

We collect demonstrations from D4RL benchmark (Fu et al., 2020). To meet our assumptions that the demonstration is limited, we only sample 50 demonstrations from the it. Not all the demonstrations are successful, there are also sub-optimal and poor demonstrations whose total rewards are below average. We accept these non-optimal demos to broaden observed state space.

B. Network architecture

We use 3 layer networks with 256, 512, 256 hidden units per layer for actor π, critic Q and generator π^{BC}. Use RELU activation for critic Q and tanh for actor π and generator π^{BC}. The generated exploration noise acts on forward propagation, and the network structure is shown in figure 1 left graph. The reference action from generator effects on back propagation and its network structure is shown in right graph. **C. Training parameters**

The discount factor γ is 0.99. The function approximators π and Q are deep neural networks with ReLU activation for Q, tanh activation for π and L2 regularization with the coefficient 10^{-4}. We use Adam optimizer with learning rate 10^{-4}. Randomly sample 640 transitions form 10 random trajectories from demonstrations and take 50000 iterations to pre-train generator. Take 750000 steps to train the agent.

For action noise, we use OU process with $\theta = 0.15, \sigma = 0.2$ to generate first part of noise. This part is weighted by a decay function $\epsilon_1 = 1 - t/6 \times 10^5$. The second part is from generator and weighted $\epsilon_2 = 1 - t/3 \times 10^5$. For loss function $T_3 = 3 \times 10^5$.

For DDPGfD, the mean total rewards for demonstrations on reply buffer is around 5000. To ensure the consistency of comparison experiment we also apply twin delay algorithm in DDPGfD.

D. Overview of experiments

We perform experiments in Walker2d, Ant and HalfCheetah Mujoco environments and do experiments with 50 and 500 demonstrations separately. We compared our method with previous works including original BC + fine tune, original TD3 and DDPGfD in section VI. For comparasion experiments the training parameters are fixed. In DDPGfD experiment we sample 10000 transitions from 10 best trajectories and loads these transitions into the replay buffer before the training begins and keeps all transitions forever.

Then we do ablation experiment to evaluate the influence of action noise, imitation loss, Q filter and hyper-parameter T_3 in section VII.

5 EXPERIMENTAL RESULT

A. Tasks
We first show the results of our method, original BC + finetune and DDPGfD on three simulated tasks:
Walker2d: Make a two-dimensional bipedal robot walk forward as fast as possible. The reward function consists forward reward, health reward and control cost.
HalfCheetah: Make a two-dimensional bipedal robot walk forward as fast as possible. The robot's legs are distributed on the 2D plane one after the other unlike Walker2d. So it's model is stable and doesn't have health reward.
Ant: Make a four-legged creature walk forward as fast as possible. Its reward function consists forward reward, health reward, contact cost and control cost.

B. Baselines
The following are 3 algorithms that form our baseline for comparisons.
BC and Finetunes: Copy the weights from the generator to the agent's policy networks and interact with the environment for fine tune.
DDPGfD: DDPGfD store the demonstration transitions in replay buffer and keep all the demonstration transitions during the training process(Vecerik et al., 2017).
Original TD3: Original TD3 without any demonstration information.

C.Results
The reward consists of four parts: forward reward (FR), health reward (HR), contact and control cost (TC and CC). In general, the larger the total reward the better the AGENT task is completed, moving quickly, reducing collisions and control consumption while not falling over. Figure 2 compares our work with original TD3 without demonstrations, Behavior Cloning + Fine tune and DDPGfD. Compared with original TD3 model, our method significantly accelerates learning speed and obtains a much better policy. It's hard for TD3 to directly learn to walk in walker2d and Ant without demonstrations. That may due to the high dimension of action space and state space. Measuring the steps costs to get to convergence, our method exhibit 2x speed up and 2x total rewards over original TD3 in Ant-v2 task.

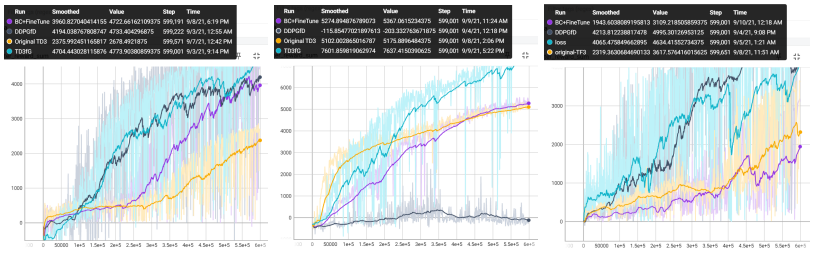

Figure 2:

Compared with DDPGfD, our method has much better performance in HalfCheetah-v2 but not as well as DDPGfD in Walker2d-v2.In the Ant-v2 task, both our method and DDPGfD worked well and their training speed do not differ much. The total reward of TD3fG is about 10% higher. However, DDPGfD does not require a pre-training process, but does require an optimised demonstration, which is not necessary for our approach. The comparative experimental results of TD3fG and DDPGfD are shown in Table I.

We believe that DDPGfD has a higher dependency on the quality of the expert presentation, as the presentation stored in the response buffer of DDPGfD will act on the entire training process. The adaptation to

Table 1: comparative experimental results

Tasks	Ant-v2	HalfCheetah-v2	Walker-v2
Original TD3	2375.99	5102.00	2319.36
Behavior Cloning	3960.82	5274.89	1943.60
DDPGfD	4194.04	-115.85	4213.81
TD3fG	4704.44	7601.85	4065.45

DDPGfD varies considerably between tasks. From the experiments we can see that, especially for Walker2d, using DDPGfD even degrades the final performance. This may be influenced by the quality and quantity of the demonstrations, with sub-optimal or non-generic demonstrations continuing to negatively affect the training process, even if these experiences have good reward value.
Overall, with the introduction of a limited number of sub-optimal demonstrations TD3fG outperforms Original TD3 and the traditional Behavior Cloning, while it requires less quality of demonstrations than DDPGfD, and its performance is relatively stable across tasks.

6 ABLATION EXPERIMENT

In this section we perform ablation experiments to evaluate the influence of various components in our method. The ablation experiments are performed on Ant-v2, Walker2d-v2 and HalfCheetah-v2. We show the following ablations on the best performing models on each task:
Q-filter + BC loss: In this method, it is up to Q filter to decide whether to introduce BC loss or not. The MSE BC loss will be imported if the Q value of reference action from generator $Q(s_t, \pi^{BC}(s_t)$ is greater than $Q(s_t, \pi(s_t))$
Decreasing weight + BC loss: This method use a linear decreasing function to control the proportion of RL loss and ME loss. The weight of the BC loss gradually decays from 3.0 to 0 in T_3 steps, while the weight of the RL loss increases from 0.2 to 1 at the same time.
BC loss + Action noise: This method add a exploration noise from the pre-trained generator which we illustrate in section IV B.
BC loss + Reply Buffer:In this method we store 10000 sub-optimal demonstration transitions in the reply buffer and test if this helps to improve the performance. This technique is similar to DDPGfD, except that it does not require the optimal demonstration to be stored in the reply buffer.

A. Q filter and Decreasing weight
The final performance and training speed in all three tasks are decreased after replacing the decreasing weights with Q filter. Figure 4 shows the training curve of TD3fG with decreasing weight and Q filter. In Ant-v2 and Walke2d-v2 the total rewards are reduced by about 20%. The first is that the Q filter does not have a smooth transition, and the second is that the critic is not reliable in the early stages of training and cannot correctly evaluate the reference movement, which weakens the role of the generator in guiding the training. These two points are not in line with the original design of the generator, so Q filter which is a common technique, is not suitable for our method. This experiment confirms that when the demonstrations are suboptimal and limited, a smooth transition between RL and IL is more helpful.

Table 2: Comparison of TD3fG and Q filter

Tasks	Ant-v2	HalfCheetah-v2	Walker-v2
TD3fG	4704.44	7601.85	4065.45
Q filter	1809.60	1631.40	3155.78

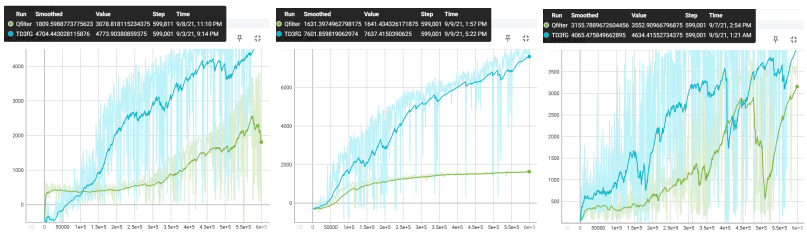

Figure 3: Comparison of TD3fG and Q filter

B. Reply Buffer

Figure 5 shows the difference between before and after using reply buffer. In Ant-v2 and Walker2d tasks, the effect of the reply buffer is negligible and in HalfCheetah-v2 the total reawrd is between DDPGfD and TD3fG. It coincides with the results of previous experiments. A major difference between Td3fG and DDPGfD is the dependence on demonstrations, while TD3fG with demonstrations in reply buffer is somewhere in between in terms of sensitivity to demonstrations quality. So for the first two tasks the performance is comparable, and in the third task the total reward lies clearly between the two methods

Table 3: Comparison of TD3fG and TD3fG with reply buffer

Tasks	Ant-v2	HalfCheetah-v2	Walker-v2
TD3fG	4704.44	7601.85	4065.45
TD3fG+ReplyBuffer	4110.68	3615.32	4047.55

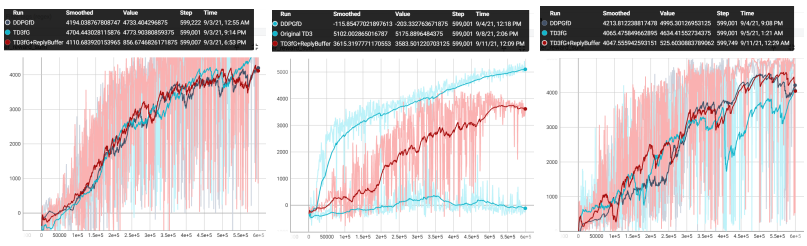

Figure 4: Comparison of TD3fG and Q filter

C. Action noise

In this part we add an extra generated action noise and compare it with TD3fG that only use BC loss. For all experiments, using reference action noise alone will boost the total reward, but not as much as just using

TD3fG with BC loss. Using both techniques at the same time does not give better results and may even be inferior to just using BC loss.

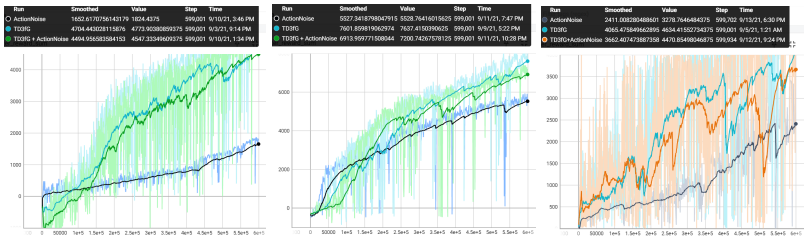

Figure 5: Comparison of TD3fG, ActionNoisea and TD3fG+ActionNoise

Table 4: Comparison of TD3fG, ActionNoisea and TD3fG+ActionNoise

Tasks	Ant-v2	HalfCheetah-v2	Walker-v2
TD3fG	4704.44	7601.85	4065.45
ActionNoise	1824.43	5527.34	2411.00
TD3fG+ActionNoise	4494.95	6913.96	3662.40

7 DISCUSSION AND FUTURE WORK

We propose an algorithm that uses pre-trained generators and dynamic weights in the loss function to overcome the detrimental effects of suboptimal, non-generalized demonstrations on the reinforcement learning training process. Unlike behavioural cloning, which is only used for pre-training, and algorithms like DDPGfD, which use demonstrations for the entire training process, our approach achieves a smooth transition from imitation learning to reinforcement learning, and performs well in simulation experiments. Our method is more general and efficient, and can be applied on any continuous control task.
We conclude from the experiments that the usefulness of behavioural cloning is limited when the number of demonstrations is relatively small, as the limited demonstrations are not sufficiently generalised and there is a cumulative error. Algorithms such as DDPGfD suffer from suboptimal demonstrations when the quality of the demonstrations is low, and techniques such as Q filter allow for selective use of demonstrations but are susceptible to ciritic influence in the early training phase.Our approach uses decreasing weights with training steps, controls the ratio of RL to BC loss, is less demanding on the demonstration, and is more generalizable to different tasks.
A limitation of our approach is that it takes time to train a generator in the first place, and the quality of the generator is critical to the final performance of the agent. This approach does not guarantee the performance of the agent in the corner case due to the limitation of the quantity of demonstrations.

8 ACKNOWLEDGEMENT

I would like to thank my colleagues for their help and advice in this project, my supervisor Prof. Ang for guiding me in my experiments and writing this paper, and my parents for their support and encouragement. Finally, my sincere thanks to Miss Clairette for her company during these months, which was crucial to me.

REFERENCES

M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. Mcgrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, and A. Ray. Learning dexterous in-hand manipulation. 2018.

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

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9329–9338, 2019.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite horizon model predictive control for nonlinear periodic tasks. *Manuscript under review*, 4, 2011.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062. PMLR, 2019.

Vinicius G Goecks, Gregory M Gremillion, Vernon J Lawhern, John Valasek, and Nicholas R Waytowich. Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments. *arXiv preprint arXiv:1910.04281*, 2019.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *Computer ence*, 2015a.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015b.

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. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, 2018. doi: 10.1109/ICRA.2018.8463162.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 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*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv e-prints*, art. arXiv:1707.08817, July 2017.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

A HYPE-PARAMETERS

Parameters	value
Width of hidden layers	(256, 512, 256)
Number of hidden layers	3
Optimizer	adam
Learning rate	0.0001
Discount factor γ	0.999
Mini-batch size	256
Actor activation function	tanh
Critic activation function	relu
Reply Buffer Size	50000
OU Process θ	0.15
OU Process σ	0.2
OU Process μ	0.0
Soft Update τ	0.001
Actor delay frequency	2
Policy noise	0.2*max aciton
Explore noise	(-0.5,0.5)
Policy decay ϵ	1/600000 (linear decay)

Table 5: Hyper-parameters of TD3

Parameters	value
Width of hidden layers	(256, 512, 256)
Number of hidden layers	3
Optimizer	adam
Learning rate	0.0001
Discount factor γ	0.999
Mini-batch size	256
Actor activation function	tanh
Critic activation function	relu
Reply Buffer Size	50000
Number of demonstrations in reply buffer	10000
OU Process θ	0.15
OU Process σ	0.2
OU Process μ	0.0
Soft Update τ	0.001
Policy noise	0.2*max aciton
Explore noise	(-0.5,0.5)
Policy decay ϵ	1/600000 (linear decay)

Table 6: Hyper-parameters of DDPGfD

Parameters	value
Width of hidden layers	(256, 512, 256)
Number of hidden layers	3
Optimizer	adam
Learning rate	0.0001
Discount factor γ	0.999
Mini-batch size	256
Actor activation function	tanh
Critic activation function	relu
Reply Buffer Size	50000
OU Process θ	0.15
OU Process σ	0.2
OU Process μ	0.0
Soft Update τ	0.001
Actor delay frequency	2
Policy noise	0.2*max aciton
Explore noise	(-0.5,0.5)
Exploration noise decay T1	1/600000
Action noise decay T2	1/300000
BC loss weight decay T3	1/300000

Table 7: Hyper-parameters of TD3

Parameters	value
Width of hidden layers	(256, 512, 256)
Number of hidden layers	3
Optimizer	adam
Learning rate	0.0001
Discount factor γ	0.999
Batch size	64 (episodes)
Randomly sampled Steps per Episode	100
Actor activation function	tanh
Number of Demonstrations	100
Training steps	100000

Table 8: Hyper-parameters of TD3

B DEMONSTRATIONS INFORMATION

Tasks	Max total reward	Min total reward	average total reward	Std
Ant-v2	5487.11	-685.92	4635.61	4635.61
Walker2d-v2	4984.58	4833.33	4926.60	33.32
HalfCheetah-v2	11035.38	10013.34	10663.37	209.35

Table 9: Hyper-parameters of TD3

C EXPERIMENT RESULTS OF DIFFERENT SEEDS

We randomly choose seeds 1, 5, 72, 141,252 to perform the experiment. The results of the experiments where the seed is 1 have been shown in the text.

C.1 ANT-V2 EXPERIMENT RESULTS

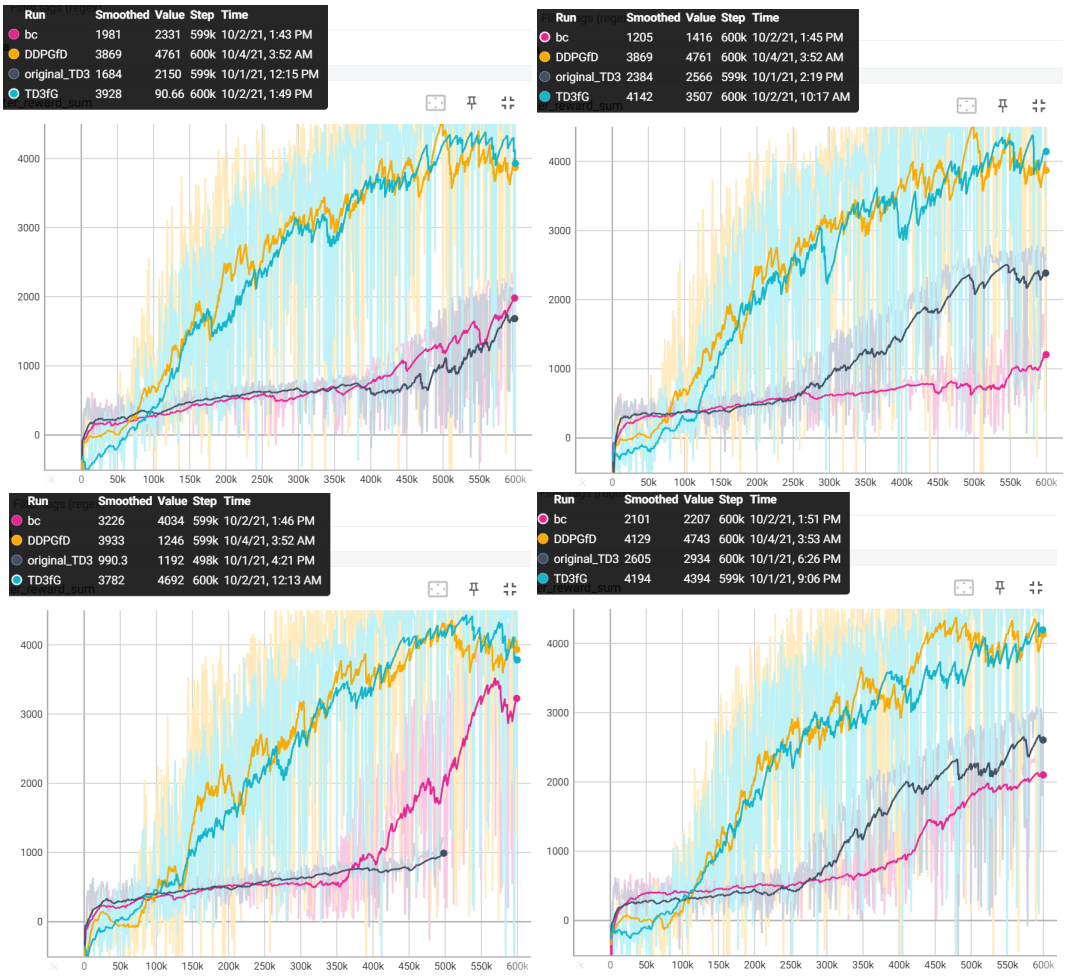

Figure 6: Ant-v2 experiment results where seeds are 5, 72, 141, 252

C.2 WALKER2D-V2 EXPERIMENT RESULTS

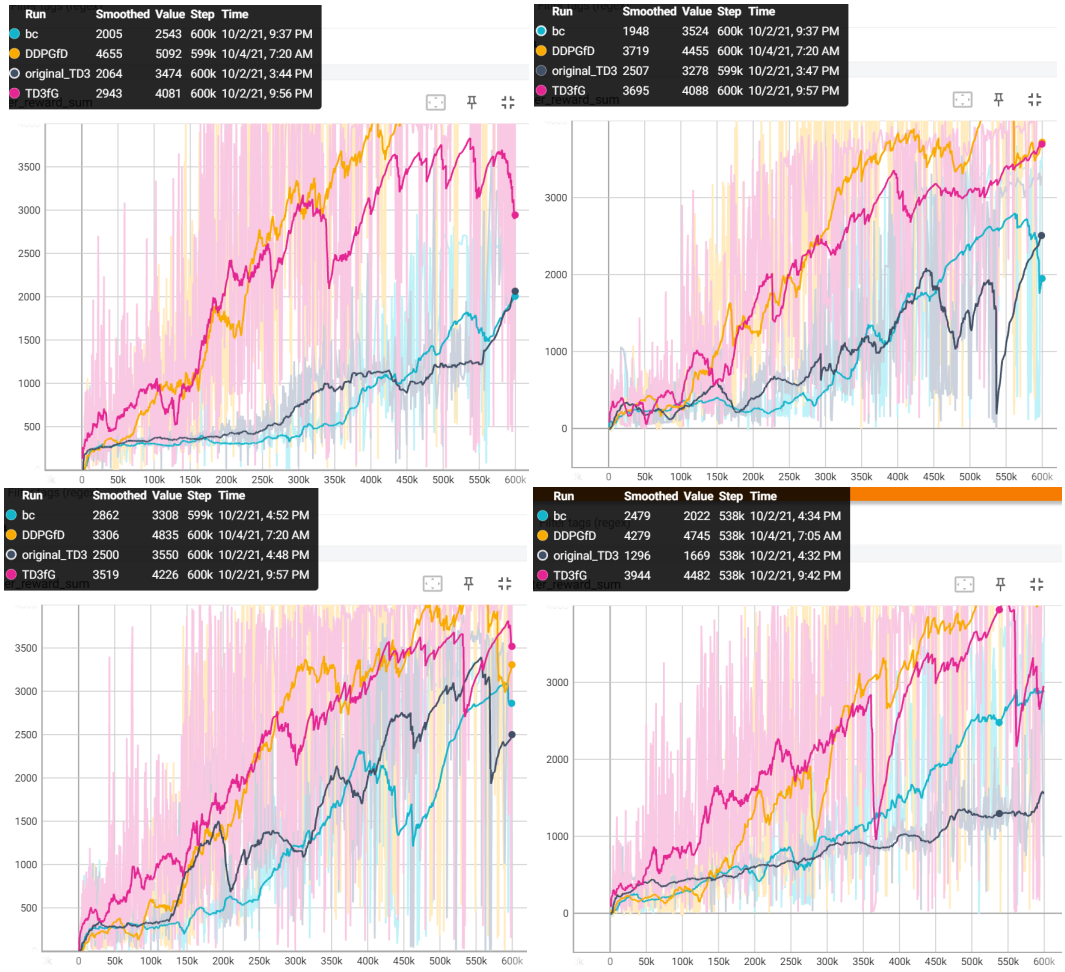

Figure 7: Walker2d-v2 experiment results where seeds are 5, 72, 141, 252

C.3 HALFCHEETAH-V2 EXPERIMENT RESULTS

Figure 8: Walker2d-v2 experiment results where seeds are 5, 72, 141, 252