
Self-Imitation Learning from Demonstrations

Appendix

George Pshikhachev
 JetBrains Research
 HSE University
 georgii39@gmail.com

Dmitry Ivanov
 JetBrains Research
 HSE University
 diivanov@hse.ru

Vladimir Egorov
 JetBrains Research
 HSE University
 vladimirrim98@gmail.com

Aleksei Shpilman
 JetBrains Research
 HSE University
 alexey@shpilman.com

1 Proofs

1.1 Proof of Lemma 1

Proof. The gradients of the loss functions L^{lb} and L_d^{lb} w.r.t. the parameters ϕ :

$$\nabla_{\phi} L^{\text{lb}} = \frac{2}{Z_{\pi}} \mathbb{E}_{s,a,R \sim \pi} [\Delta_{\pi}^+]^2 \nabla_{\phi} Q_{\phi}(s, a)$$

$$\nabla_{\phi} L_d^{\text{lb}} = 2 \frac{1-\lambda}{Z} \mathbb{E}_{s,a,R \sim \pi} [\Delta_{\pi}^+]^2 \nabla_{\phi} Q_{\phi}(s, a) + 2 \frac{\lambda}{Z} \mathbb{E}_{s,a,R \sim \pi_E} [\Delta_{\pi_E}^+]^2 \nabla_{\phi} Q_{\phi}(s, a)$$

The difference of the gradients $\nabla_{\phi} L_{\text{diff}}^{\text{lb}} = \nabla_{\phi} L_d^{\text{lb}} - \nabla_{\phi} L^{\text{lb}}$ after rearranging terms can be written as:

$$\nabla_{\phi} L_{\text{diff}}^{\text{lb}} = 2 \frac{\lambda}{Z} [\mathbb{E}_{s,a,R \sim \pi_E} [\Delta_{\pi_E}^+]^2 \nabla_{\phi} Q_{\phi}(s, a) - \frac{Z_{\pi_E}}{Z_{\pi}} \mathbb{E}_{s,a,R \sim \pi} [\Delta_{\pi}^+]^2 \nabla_{\phi} Q_{\phi}(s, a)]$$

If $|\nabla_{\phi} L_{\text{diff}}^{\text{lb}}| > 0$, then $|\nabla_{\phi} L_d^{\text{lb}}| > |\nabla_{\phi} L^{\text{lb}}|$, i.e. adding demonstrations to the buffer increases the gradient's l1-norm. The condition $|\nabla_{\phi} L_{\text{diff}}^{\text{lb}}| > 0$ can be written as:

$$\mathbb{E}_{s,a,R \sim \pi_E} [\Delta_{\pi_E}^+]^2 |\nabla_{\phi} Q_{\phi}(s, a)| - \frac{Z_{\pi_E}}{Z_{\pi}} \mathbb{E}_{s,a,R \sim \pi} [\Delta_{\pi}^+]^2 |\nabla_{\phi} Q_{\phi}(s, a)| > 0$$

After moving the second term to the right-hand side of the inequality, dividing both sides by Z_{π_E} , and applying definitions of Z_{π} and Z_{π_E} , we finally get:

$$\frac{\mathbb{E}_{s,a,R \sim \pi_E} [\Delta_{\pi_E}^+]^2 |\nabla_{\phi} Q_{\phi}(s, a)|}{\mathbb{E}_{s,a,R \sim \pi_E} \Delta_{\pi_E}^+} > \frac{\mathbb{E}_{s,a,R \sim \pi} [\Delta_{\pi}^+]^2 |\nabla_{\phi} Q_{\phi}(s, a)|}{\mathbb{E}_{s,a,R \sim \pi} \Delta_{\pi}^+}$$

□

1.2 Proof of Lemma 2

Proof. From the definition of completely useless demonstrations, it follows that:

$$Z_{\pi_E} = \mathbb{E}_{s,a,R \sim \pi_E} \Delta_{\pi_E}^+ = 0$$

and

$$\mathbb{E}_{s,a,R \sim \pi_E} [\Delta_{\pi_E}^+]^2 = 0$$

It follows that both terms of $\nabla_{\phi} L_{\text{diff}}^{\text{lb}}$ equal 0:

$$\nabla_{\phi} L_{\text{diff}}^{\text{lb}} = \nabla_{\phi} L_d^{\text{lb}} - \nabla_{\phi} L^{\text{lb}} = 2 \frac{\lambda}{Z} [\mathbb{E}_{s,a,R \sim \pi_E} [\Delta_{\pi_E}^+]^2 \nabla_{\phi} Q_{\phi}(s, a) - \frac{Z_{\pi_E}}{Z_{\pi}} \mathbb{E}_{s,a,R \sim \pi} [\Delta_{\pi}^+]^2 \nabla_{\phi} Q_{\phi}(s, a)] = 0$$

Thus, adding completely useless demonstrations to the buffer has no effect on the gradient of the loss function w.r.t. the parameters:

$$\nabla_{\phi} L_d^{\text{lb}} = \nabla_{\phi} L^{\text{lb}}$$

□

1.3 Proof of Lemma 3

Proof. Using the definition of \bar{p} , the condition for it to decrease can be written as:

$$\frac{\mathbb{E}_{s,a,R \sim \pi_E} \Delta_{\pi_E}^+}{\mathbb{E}_{s,a,R \sim \pi} \Delta_{\pi}^+} > \frac{\mathbb{E}_{s,a,R \sim \pi_E} (\Delta_{\pi_E} - Q_{\text{diff}})^+}{\mathbb{E}_{s,a,R \sim \pi_{new}} (\Delta_{\pi_{new}} - Q_{\text{diff}})^+}$$

where left-hand side and right-hand side represent $\frac{\bar{p}}{1-\bar{p}}$ before and after the update. After rearranging:

$$\mathbb{E}_{s,a,R \sim \pi_{new}} (\Delta_{\pi_{new}} - Q_{\text{diff}})^+ > \frac{\mathbb{E}_{s,a,R \sim \pi_E} (\Delta_{\pi_E} - Q_{\text{diff}})^+}{\mathbb{E}_{s,a,R \sim \pi_E} \Delta_{\pi_E}^+} \mathbb{E}_{s,a,R \sim \pi} \Delta_{\pi}^+$$

After subtracting $\mathbb{E}_{s,a,R \sim \pi} \Delta_{\pi}^+$ from both sides of the inequality, this yields:

$$d(\pi, \pi_{new}) > \left(\frac{\mathbb{E}_{s,a,R \sim \pi_E} (\Delta_{\pi_E} - Q_{\text{diff}})^+}{\mathbb{E}_{s,a,R \sim \pi_E} \Delta_{\pi_E}^+} - 1 \right) \mathbb{E}_{s,a,R \sim \pi} \Delta_{\pi}^+$$

□

2 Limitations of the proposed algorithm

In contrast to well-studied algorithms like DQN, PPO, or DDPG, SIL is a relatively novel method that has not been tested as extensively and that still may exhibit certain engineering issues. One of such issues is overestimation of the lower-bound value by critic, which can happen for several reasons. First, stochasticity of policy or environment dynamics can cause high variance of return distribution in a given state, making critic estimate the highest rather than the expected return. This can be partially mitigated by using generalized SIL with n-step update [2]. Second, since value approximations for all states are conditioned on the same vector of parameters ϕ , updating ϕ to increase $V_{\phi}(s_1)$ can increase $V_{\phi}(s_2)$ as well, even if the latter is already tight. Third, naive initialization of ϕ can cause critic to overestimate the lower bounds in some states before the training even begins. While value overestimation is not specific to SILfD, it is partially alleviated in the original SIL by alternating with on-policy updates that can decrease overestimated values. In contrast, the issue can be exaggerated in

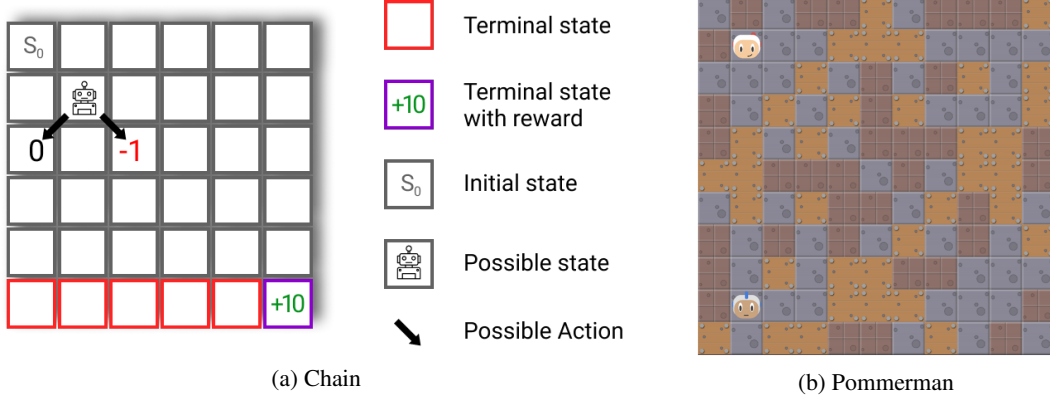


Figure 1: Chain and Pommerman environments

SILfD: if demonstrations contain states that the agent does not reach, the value estimates of these states may rise uncontrollably.

A distinctive feature of SILfD is prioritization of demonstrations with high returns. While this feature makes SIL robust to suboptimal demonstrations, it can also backfire if demonstrations contain useful behaviour that does not reach any reward. For example, consider the task of stacking three cubes by a robot manipulator. If a demonstration of successfully stacking three cubes is provided, it will be used by SILfD to recover the demonstrated behaviour. However, if a demonstration of only stacking two cubes is provided and no reward is achieved, it will be deemed useless and ignored by SILfD.

3 Environments

Here we report some technical details on environments.

3.1 Chain

In Chain Environment, observation space is represented by a square grid. Agent’s state in the environment is defined by its position on this grid and consists of two components: horizontal and vertical. Each component is normalized to be in range from -1 to 1 . Thus, the initial state in the environment is $[-1, -1]$, and for all the terminal states the second component is 1 . The goal of the agent is to reach state $[1, 1]$, i. e. the bottom right corner of the grid. Action space is discrete and specifies two options: going left (0) and going right (1). The total reward for the episode lies within the range $[-60, 100]$.

All experiments were conducted on CPUs of MacBook Pro 2017 with 8 GB 2133 MHz LPDDR3 RAM. Each algorithm was trained for 500 thousands transitions in the environment, which constitutes up to 30 minutes of wall-clock time, depending on the algorithm. The exceptions are DQfD, which is more sample efficient and was trained for 100 thousands transitions, and BC and DT, which do not require interacting with the environment. For SILfIL, we collect 1000 BC demonstrations to fill the buffer of SIL.

3.2 DeepMind Control Suite

All experiments were conducted on Intel Core i7 2.5 GHz CPU of MacBook Pro 16 with 16 GB 1600 MHz DDR3 RAM.

In the original Cartpole, a non-zero reward is given every time the pole is positioned vertically and the cart is located within a certain range. In our modification, the agent does not receive the reward if the velocities of the pole or the cart exceed a threshold. As a result, it becomes extremely difficult for the agent to find learning signal randomly. In order to collect demonstrations, we handcraft a heuristic PID-controller that swings the cart back and forth until the pole reaches a vertical position. In the locomotive tasks, the reward is originally given each time the robot advances forward. In our modifications, the agent receives the reward proportional to its velocity, but only if the velocity

exceeds a certain threshold. As an expert, we train a PPO agent to move with a target speed that is only slightly higher than the threshold. Since it is possible to move faster than such expert, demonstrations of its behaviour are suboptimal for the agent.

For SILfIL, we collect 25 BC demonstrations to fill the buffer of SIL in each DMC environment. Note that we filter these demonstrations and only select those with non-zero return.

3.2.1 Cartpole

The state space is defined by 5 components: position of the cart, cosine of the pole, sine of the pole and cart’s and pole’s velocities. Action space is continuous and one-dimensional. Action’s magnitude specifies the amount of force applied to move a cart and the sign of the action specifies the direction of the force. In the original implementation of the environment the reward is given whenever the horizontal position of the cart is within a range of from -0.25 to 0.25 and the pole’s cosine is within a range from 0.995 to 1 . In our modification agent does not receive any reward whenever the pole’s and cart’s velocities exceed 0.25 and 0.5 respectively. To collect demonstrations in this environment, we handcrafted a heuristic PID-controller. It receives 13 total reward per episode on average. However, we filtered demonstrations containing only those episodes where the total reward exceeded 40, and thus the expert’s performance in demonstrations is approximately 65.

3.2.2 Walker

In the Walker environment, the agent controls the robot that has two actuated legs. The action space consists of 6 components, where each component represents the force applied to a certain joint of a single leg. The state space consists of 24 components that represent positions, orientations and velocities of different parts of the robot. In the original implementation of the environment the reward is divided into two parts: the first part is the standing reward r_s which increases towards 1 when the vertical position of the agent’s torso gets closer to the value of 1.2; the second part is the moving reward r_m which increases up to 1 as the agent’s horizontal velocity gets closer and exceeds the threshold of 8. The final reward is calculated using the following equation:

$$r = r_s \frac{(5r_m + 1)}{6} \quad (1)$$

In order to collect demonstrations for experiments in Walker, we modify the moving part of the original reward. To train the agent run with a certain target speed, we set the reward to be 1 if the velocity of the agent lies inside the interval between 4 and 5 and we decrease the reward as it gets further from its boundaries. We train agent to maximize this reward using PPO algorithm.

To create sparse version of the environment, we modify the moving reward to be 0 if the agent’s velocity is below the threshold of 4. In order to encourage the agent to run faster as it exceeds this threshold, we linearly increase the reward depending on agent’s velocity.

3.2.3 Hopper

In the Hopper environment, the agent controls the robot that has a single leg. The state and action spaces are represented by the vectors with 15 and 4 components respectively. The reward scheme is the same as in the Walker Environment, but the target speed for the expert lies in the interval between 1 and 2. In the sparse version of the environment the speed threshold is set to be 1.3.

3.2.4 Cheetah

In the Cheetah environment the agent controls the robot with two legs: back and front. The state and action spaces are represented by the vectors with 17 and 6 components respectively. The reward scheme is similar to Hopper’s and Walker’s reward scheme, but the standing reward is absent. The target speed for the expert lies in the interval between 5 and 6, and the velocity threshold in sparse version of the environment is set to be 5.

3.3 Pommerman

The observation space is composed of a 11x11 grid with 15 one-hot features, 2 feature maps, and an additional information vector. The one-hot feature represents an element on the map. Specifically, these features can represent the current player, an ally, an enemy, a passage, a wall, wood, a bomb, flame, fog, and a power-up. The feature maps contain integers indicating bomb blast strength and bomb life for each location. Finally, the additional information vector contains the time-step, number of ammunition, whether the player can kick and blast strength for the current player. The agent has six actions: do-nothing, up, down, left, right, and lay bomb. We use architecture identical to [1] for all neural networks.

We use Agent47Agent to gather expert trajectories, which is based on Monte-Carlo Tree Search.¹ We only select winning trajectories from the expert.

All experiments were conducted on one Tesla V100 GPU. Each algorithm was trained for 10 million transitions in the environment. The exceptions are DQfD, which is more sample efficient and was trained for 1 million transitions, and BC, which does not require interacting with the environment.

For SILfIL, we collect 400 BC demonstrations to fill the buffer of SIL. These are not filtered.

4 Hyperparameters

Below we describe how hyperparameters were tuned. The selected hyperparameters are reported in Table 1.

In all environments and for each algorithm, we select hyperparameters using a random search procedure. Specifically, we select a 100 random hyperparameter configurations, train an algorithm with each configuration on 2 random seeds, evaluate the trained algorithm for 100 episodes, and select the configuration with the highest average score over 2 seeds and 100 episodes. We then rerun each experiment 3 times (DMC, Pommerman) or 5 times (Chain) with the selected hyperparameters, the results of which are reported in the plots.

In Chain, we select a configuration of hyperparameters that achieves the highest score in solving the task with one optimal and one completely useless demonstrations provided (1 suboptimal demonstration on the x-axis of the plot in the main text) and apply these hyperparameters to the other tasks. In DMC, we select hyperparameters on Half Cheetah environment and apply these hyperparameters in Walker, Hopper, and Cartpole. Since experiments in Pommerman require more compute, the number of tested hyperparameter configurations is reduced to 25, and some hyperparameters (batch size, epochs per update, some GAIL hyperparameters, as well as order, number, and size of hidden layers) are fixed without tuning as the hyperparameters proposed in [1].

We select hyperparameters for the following algorithms: BC, SILfD, POfD, DQfD / DDPGfD. SILfIL and IL \rightarrow SIL use the same hyperparameters as in SILfD and BC. For Decision Transformers, we use the hyperparameters reported in the original paper for D4RL (which contains analogues of environments in DMC). We also simplify the architecture of DT in Chain.

For all algorithms, we select learning rate from $\{2e-5, 1e-4, 2e-4, 5e-4, 1e-3\}$ and batch size from $\{64, 256, 512\}$. For all algorithms but BC, we select epochs per update from $\{3, 10, 30\}$. For POfD, we also select discriminator batch size from $\{128, 256, 512\}$, discriminator epochs per update from $\{3, 10\}$, discriminator learning rate from $\{1e-6, 1e-5, 1e-4, 5e-4\}$, and reward mixing coefficient $\lambda_1 \in \{0.01, 0.1, 1, 10, 100\}$. For SILfD, we also select epochs per SIL update from $\{5, 10, 20, 40\}$, SIL loss weight from $\{0.01, 0.1, 1, 10\}$, SIL value loss weight β from $\{0.01, 0.1, 1\}$, SIL batch size from $\{256, 512\}$. Online updates in SIL are based on PPO in all environments. For DQfD and DDPGfD, we also select n-step loss weight from $\{0.01, 0.1, 1\}$, l2 regularization weight from $\{1e-5, 1e-4, 1e-3\}$, priority bonus of demonstrations from $\{0.2, 0.5, 1\}$, supervised loss weight from $\{0.01, 0.1, 1\}$. We apply dueling [4] and double [3] modifications of DQN in DQfD. We train DQfD and DDPGfD 5 times as less as other algorithms to utilize their sample efficiency and prevent overfitting.

¹Code: <https://github.com/YichenGong/Agent47Agent/tree/master/pommerman>

References

- [1] P. Barde, J. Roy, W. Jeon, J. Pineau, C. Pal, and D. Nowrouzezahrai. Adversarial soft advantage fitting: Imitation learning without policy optimization. *arXiv preprint arXiv:2006.13258*, 2020.
- [2] Y. Tang. Self-imitation learning via generalized lower bound q-learning. *arXiv preprint arXiv:2006.07442*, 2020.
- [3] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
- [4] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

Table 1: Hyperparameters for all environments and algorithms. PPO is not a separate baseline, but POfD and on-policy component of SIL are based on POfD are based on PPO.

Hyperparameter	Chain	Pommerman	DMC
PPO			
Batch size	32	256	512
GAE λ	0.95	0.95	0.95
Entropy coefficient	0.01	0.01	0
Transitions between updates	1000	128	1000
Number of workers	1	8	8
Epochs per update	3	3	10
Update clipping parameter	0.2	0.2	0.2
Learning rate	2e-4	2e-4	2e-4
γ	0.99	0.99	0.99
SIL			
SIL value loss weight β	0.01	0.1	0.1
Epochs per SIL update	40	10	5
SIL loss weight	10	0.01	1
SIL Batch size	256	512	256
Buffer size	1e5	1e5	1e5
POfD			
Learning rate	2e-5	2e-4	2e-4
Mixing coefficient λ_1	10	1	0.1
Discriminator learning rate	1e-5	1e-6	1e-4
Discriminator batch size	32	256	512
Discriminator epochs per update	10	10	10
DQfD and DDPGfD			
Buffer size	1e5	1e6	1e5
Learning rate	2e-4	5e-4	1e-4
Demos priority bonus ϵ_d	0.5	0.2	0.2
Number of steps in n-step loss	10	10	10
N-step loss weight λ_1	0.01	1	0.1
Supervised loss weight λ_2	0.1	1	-
l2 regularization weight	0.001	0.001	1e-5
BC			
Learning rate	1e-3	2e-4	1e-3
Batch size	32	64	512
Number of batches	4096	250 000	12288
DT			
Learning rate	1e-3	1e-4	1e-4
Batch size	32	64	64
Number of batches	20000	100000	100000
Number of attention layers	1	3	3
Dropout rate	0	0.1	0.1