FASTER REINFORCEMENT LEARNING WITH VALUE TARGET LOWER BOUNDING

Anonymous authors
Paper under double-blind review

ABSTRACT

We show that an arbitrary lower bound of the optimal value function can be used to improve the Bellman value target during value learning. In the tabular case, value learning using the lower bounded Bellman operator converges to the same optimal value as using the original Bellman operator, at a potentially faster speed. In practice, discounted episodic return from the training experience or discounted goal return from hindsight relabeling can serve as the value lower bound when the environment is deterministic. This is because the empirical episodic return from any state can always be repeated through the same action sequence in a deterministic environment, thus a lower bound of the optimal value from the state. We experiment on Atari games, FetchEnv tasks and a challenging physically simulated car push and reach task. We show that in most cases, simply lower bounding with the discounted episodic return performs at least as well as common baselines such as TD3, SAC and Hindsight Experience Replay (HER). It learns much faster than TD3 or HER on some of the harder continuous control tasks, requiring minimal or no parameter tuning.

1 INTRODUCTION

In Temporal Difference (TD) learning, the value function is adjusted toward its Bellman target, which is the reward of the current step plus the discounted value of the next state. This forms the basis of many state of the art reinforcement learning (RL) algorithms such as DDPG (Lillicrap et al., 2015), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018).

The value of the next state is typically estimated using "bootstrapped values" which follow the learned values in a delayed schedule. During the initial phases of training, the bootstrapped values are usually random, due to the randomly initialized values, and can be far away from the optimal values. Consequently, the Bellman value targets as well as the learned values are usually far away from the optimal values.

This brings us to the following idea: If we can provide a value target that's closer to the optimal value, can we speedup TD learning without harming its convergence properties. For example, we know that the optimal value is just the expected discounted return of the optimal policy, which always upper bounds the expected return of any policy, including the behavior policies used to collect the training trajectories. For episodic RL tasks, we could potentially use the observed discounted return up to episode end from the training trajectories to lower bound the value target. This makes the new value target closer to the optimal value, when the empirical discounted return is higher than the Bellman target.

Now the question is, will such a way of lower bounding the value target work: First, will it still converge? Second, will it converge to the optimal value? Third, will it speed up value learning?

2 THEORETICAL RESULTS FOR THE TABULAR CASE

For the tabular case, value target lower bounding seems to converge to the same optimal value as the original Bellman value learning, and the proof also seems straightforward.

2.1 BACKGROUND

In finite MDPs with a limited number of states and actions, a table can be used to keep track of the value of each state during learning. Using dynamic programming algorithms such as value iteration, starting from any initialization, the state values are guaranteed to converge to the optimal value through Bellman updates (Chapter 4.4 in Sutton & Barto (2018)).

Algorithm 1: Bellman value iteration

Data: Finite MDP $p(s', r|s, g, a)$, convergence threshold θ
Result: State value $v(s)$

```
v(s) ← 0 repeat
    Δ ← 0;
    for each state s do
        v ← v(s);
        v(s) ← max_a Σ_{s',r} p(s', r|s, g, a)[r + γv(s')];
        Δ ← max(Δ, |v(s) − v|);
    end
until Δ < θ;
```

The core of the algorithm is the Bellman update of the value function, $\mathcal{B}(v)$:

$$\mathcal{B}(v)(s) \coloneqq \max_a \sum_{s',r} p(s', r|s, g, a)[r + \gamma v(s')] \tag{1}$$

It is well known that the Bellman operator, $\mathcal{B}$, is a contraction mapping over value functions (Denardo, 1967). That is, for any two value functions v_1 and v_2, $|\mathcal{B}(v_1) - \mathcal{B}(v_2)| \leq \gamma|v_1 - v_2|$ for the discount factor $\gamma \in [0, 1)$. This guarantees that any value function under the algorithm converges to the optimal value.[1]

2.2 VALUE TARGET LOWER BOUNDING CONVERGENCE THEOREM

Theorem 1. *Suppose the optimal value under the Bellman operator is $\mathcal{B}^\infty(v)$. For any value function f that lower bounds the optimal value, i.e. $\forall s, f(s) \leq \mathcal{B}^\infty(v)(s)$, if we define the lower bounded Bellman operator as $\mathcal{M}_f \circ \mathcal{B}(v) \coloneqq \max(\mathcal{B}(v), f)$, then $(\mathcal{M}_f \circ \mathcal{B})^\infty(v)$ converges to $\mathcal{B}^\infty(v)$.*

A few things to note about the proof (see Appendix A.1).

First, this only proves convergence, not contraction under the original $||v_1 - v_2||_\infty$ metric. In the case of the Bellman operator, contraction shows that $\forall v_1, v_2$ value functions, $||\mathcal{B}(v_1) - \mathcal{B}(v_2)||_\infty \leq \gamma||v_1 - v_2||_\infty$. Here, for value target lower bounding, there can be counter examples where $\mathcal{M}_f \circ \mathcal{B}$ does not always contract in the original metric space for value functions. Here, convergence relies on the convergence of the Bellman value iteration and the existence of the fixed point v^*. One difficulty caused by this change is that the stopping criterion in Algorithm 1 ($\Delta < \theta$) no longer works, as we do not have access to the converged value during learning. This is perhaps not a serious concern in practice, as people often train algorithms for a fixed number of iterations or time steps.

Second, in terms of convergence speed, the new algorithm is at least as fast as the original, and the closer the lower bound f is to the optimum value v^*, the more likely the algorithm converges faster, because it's more likely $f(s) > \mathcal{B}(v_1)(s)$. It is strictly faster if f helps close the widest gaps between v_1 and v^* in the state space. At the beginning of training, the widest gaps are maybe around the optimal trajectories and close to high reward regions.

Third, the lower bound function doesn't have to be static during training. As long as there is a single f during each iteration, convergence property is preserved.

For example, the lower bound function can be obtained from off-policy data collected during training. In off-policy algorithms, f can come from rewards obtained along trajectories in the replay

[1] See, for example, `https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/lecture-notes/lecture5-2pp.pdf`, page 8 for the gist of the proof.

buffer. As long as the environment is deterministic, the empirical returns can always be achieved following the same action sequence, and will always lower bound the optimal value.

3 EXAMPLE LOWER BOUND FUNCTIONS

We show a few cases where lower bound functions can be readily obtained from the training experience. There are probably many other ways to obtain useful lower bounds, which we encourage you, the reader, to come up with.

3.1 EPISODIC TASKS

In episodic tasks, discounted return is only accumulated up to the last step of an episode. In this case, we can wait until an episode ends, and compute future discounted returns of all time steps inside the episode. This discounted return is guaranteed to be a lower bound of the optimal value, if the environment is deterministic, i.e. the reward sequence can be repeated using the exact same sequence of actions. To make training efficient, we can compute and store such discounted returns into the replay buffer for each time step, and simply read them out during training.

We call this variant lb-DR, short for lower bounding with discounted return.

3.1.1 EPISODIC WITH HINDSIGHT RELABELED GOALS

In goal conditioned tasks, one helpful technique is hindsight goal relabeling (Andrychowicz et al., 2017). It takes a future state that is d time steps away from the current state as the hindsight / relabeled goal for the current state. When the goal is reached, a reward of 0 is given, otherwise a -1 reward is given for each time step.

In this case, we know it took d steps to reach the hindsight goal, so the discounted future return is:

$$\begin{aligned} R_d &= \sum_{i=0,..,d-1} -1\gamma^i \\ &= -1(1-\gamma^d)/(1-\gamma) \end{aligned} \tag{2}$$

This calculation can be done on the fly as hindsight relabeling happens, requiring no extra space and very little computation.

We call this variant lb-GD, short for lower bounding with goal distance based return.

Additionally, we can also apply lb-DR and lb-GD together, with discounted return lower bounding (lb-DR) on the original experience and goal distance return lower bounding (lb-GD) on the hindsight experience, giving the lb-DR+GD variant.

3.2 NON-EPISODIC TASKS WITH POSITIVE REWARDS

When the task is continuing, without any episode end, discounted return needs to be accumulated all the way to infinity. This makes it difficult to lower bound the value if rewards can be negative. When rewards are always non-negative, one can still use the discounted return of the future n-steps to lower bound the value. See Chapter 3.3 of (Sutton & Barto, 2018) for details on episodic vs continuing tasks.

4 INTEGRATION INTO RL ALGORITHMS

4.1 BACKGROUND

The value target lower bounds can be readily plugged into RL algorithms that use TD return as the value target, e.g. DDPG or SAC.

In these algorithms, the action value function $q(s, a)$ is learned through a squared loss with the target action value $\bar{q}$. In one step TD return, for a batch $\mathbf{B}$ of transitions $\{s, a \rightarrow r, s'\}$ sampled from the replay buffer, the loss is defined as:

$$\mathcal{L}_q \coloneqq \sum_{(s,a,r,s') \in \mathbf{B}} |q(s,a) - y|^2 \tag{3}$$

where y is the value target, and in the case of one step TD return, y is assigned the one step TD return $\hat{q}(s, a, r, s')$ defined as the reward plus the discounted bootstrap value of the next state:

$$\hat{q}(s, a, r, s') \coloneqq r + \gamma q'(s', \mu'(s')) \tag{4}$$

Here, q' and μ' are the bootstrap value and policy functions, typically following the value and policy functions in a delayed schedule during training. (They are also called "target value" and "target policy", and are very different from the "value target" in this paper.)

4.2 Value Target Lower Bounding

With lower bounding, we can simply replace the value target y with the lower bounded value target:

$$y \leftarrow \max(f, \hat{q}(s, a, r, s')) \tag{5}$$

This way of simply lower bounding value target does not require any parameter, but one can always interpolate between these two value targets using a mixing weight α:

$$y \leftarrow (1 - \alpha)\hat{q}(s, a) + \alpha \max(f, \hat{q}(s, a)) \tag{6}$$

A small α dampens the effect of the new value target, and may be desirable in practice when assumptions of the theorem could be violated, e.g. in function approximation or in non-deterministic environments.

5 Experiments

The goal here is to demonstrate that lower bounding the value target can improve over baseline methods such as DDPG, TD3, SAC and HER. Because the lower bounded value target can now look potentially many steps into the future, we suspect it to be best suited for long horizon, sparse reward tasks. We picked the following tasks, experimented with only these tasks, and reported results on all the tasks that we experimented on.

5.1 Environments and Tasks

We experiment on three sets of tasks with different input characteristics and control difficulty. Some of the tasks are not goal conditioned, so only lower bounding with empirical discounted return is available. Some of them are goal conditioned, so both empirical discounted return and hindsight relabeling with discounted goal return as lower bound are available.

5.1.1 Atari games

We experiment on the classical Atari games with image input to test using discounted episodic return to lower bound value target. We randomly picked the games Breakout, Seaquest and SpaceInvaders, because they have relatively sparse reward, and only carried out experiments on them. We evaluate on the deterministic versions of the three games.

The two games Seaquest and Space Invaders turn out to be "pseudo-deterministic": they utilize internal timers and/or player actions to decide when to introduce game objects into the scene[2]. This makes the game partially observable and in effect non-deterministic to the RL agent. We tune parameter and report results for them separately.

[2] https://computerarcheology.com/Arcade/SpaceInvaders/

5.1.2 EPISODIC FETCH PUSH, SLIDE AND PICKANDPLACE

The FetchEnv tasks are goal conditioned tasks with a robotic arm moving objects on a table. Robot and object states serve as input and continuous actions as policy output, with the actions taking the form of relative positions to be fed into the PID controller of the robot.

By default the FetchEnv tasks are non-episodic. They reset every 50 steps, but all steps including the step right before task reset have the same positive discount (Andrychowicz et al., 2017). To make them episodic, we add a gym wrapper around the environment to end an episode after its goal is achieved, and reset the task. When a goal is not reached within 50 steps, we just reset the task without ending the episode, as in the original FetchEnv, and such experience are not used in value target lower bounding. Rewards are sparse and goal-conditioned, with -1 for non-goal states and 0 for goal states.

Compared with the Atari games, the inputs are simpler, no longer image based, but the control task is continuous, under realistic physical simulation and harder.

5.1.3 PIONEER PUSH AND REACH TASKS

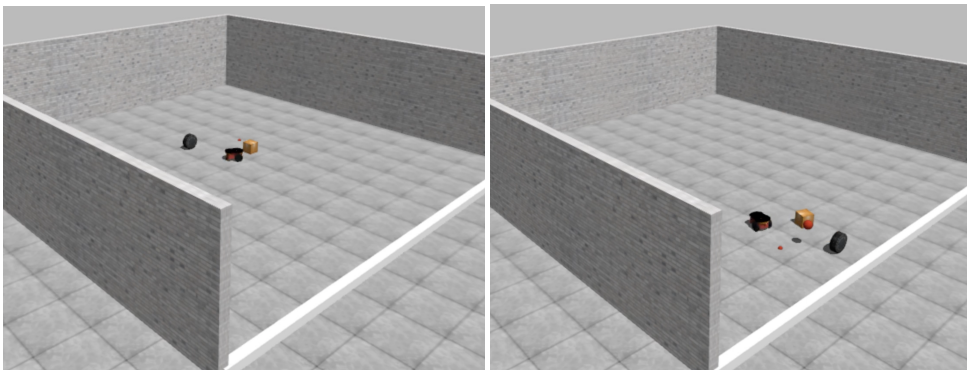

Figure 1: The Pioneer Push task and the Push and Reach task.

This is a set of challenging goal reaching and object pushing tasks for the physically simulated car Pioneer 2dx. The car is 0.4 meter long. Objects and goal positions are randomly initialized between 0.5 meter to 1 meter of each other inside a 10 meter by 10 meter flat space. Inputs are the car and object states and the goal positions, and actions are the forces applied on the two driving wheels.

For the Pioneer Push task, the car has to push a block to within 0.5 meter of the 2 dimensional goal position indicated by a small red dot on the ground. For the Pioneer Push and Reach task, the car has to first push the object to the goal location (red dot) and then drive to a separate goal position (red ball in the air); the goal is achieved when the concatenation of the two goal locations (for Push and for Reach) is within 0.5 of the concatenated achieved positions (of the block and the car) in L_2-distance.

Similar to FetchEnv, we make the tasks episodic with sparse goal reward.

These tasks take longer time to accomplish, and also take longer time to train than the FetchEnv tasks. Some of the reasons are the indirect wheel control instead of position control, and the arena space being much larger than just a tabletop.

5.2 BASELINES

Baselines include DDPG (Lillicrap et al., 2015), TD3 (Fujimoto et al., 2018), SAC (Haarnoja et al., 2018) and HER (Andrychowicz et al., 2017). Implementations are based on open sourced repositories, and baseline performance is verified against published results.

5.3 HYPERPARAMETERS

The value target lower bounding method has only one parameter, the mixing weight, for tuning. We did not find it necessary to tune it on Atari Breakout, FetchEnv or Pioneer tasks, and had to tune it on the pseudo deterministic Atari games, using 0.0005 for both games. We did not re-tune any of the parameters of the baselines.

5.3.1 HYPERPARAMETERS FOR BASELINES

Hyperparameters of the baseline algorithms follow published work in the case of FetchEnv (Plappert et al., 2018). For Atari and Pioneer Push and Reach tasks, they are tuned using one set of random seeds and after keeping the hyperparameters fixed, trained with a different set of random seeds and evaluated during training.

For the Atari games with SAC baseline, there are no strong reported results, so we tuned the hyperparameters a bit and found it to outperform published Actor-Critic results on Atari Breakout (Oh et al., 2018). We use 30 environments, unrolling 8 time steps every training iteration, with each iteration containing 4 updates each with a minibatch of 500 transitions sampled from the 2.4 million time step replay buffer. 500 time steps are collected before training starts. Discount $\gamma = 0.99$. The SAC target entropy is set to the entropy of uniformly distributing 0.1 probability mass across all but one actions. Actions are repeated deterministically for 4 frames (3 for Space Invaders as in DQN (Mnih et al., 2013)), and the most recent 4 frames are stacked and rescaled to [-1, 1] to form the 84x84x4 input tensor. Rewards are clipped between -1 and 1. Network structures are the same as Double DQN (van Hasselt et al., 2015) with 3 convolution layers, with input layer 32 filters of size 8 stride 4, then 64 filters of size 4 stride 2 and 64 filters with size 3 stride 1, 1 fully connected of size 512 before output. We train for 12 million steps (48 million frames) for each task and evaluate every 1000 iterations averaging across 20 episodes using ϵ-greedy policy with 5% random actions.

For FetchEnv tasks, DDPG and HER learn faster than their TD3 variants and are reported here. Hyperparameters are the same as used by Plappert et al. (2018), with 38 parallel environments unrolling 50 time steps per train iteration, training 40 updates per iteration, targets are updated once every 40 updates. For each update, a minibatch of 5000 transitions are sampled from the replay buffer of size 2 million. Discount $\gamma = 0.98$. Actions are ϵ-greedy with 30% random actions. 80% hindsight experience. Observations are normalized to have zero mean and unit variance based on the statistics of the recent observations.[3] Networks are 3 fully connected layers of size 256. Length of the episodes are capped at 50. We train for 2 million frames and evaluate every 40 iterations averaging across 200 episodes.

For Pioneer Push and Reach tasks, TD3 is used, (we simply equip DDPG with two critics for clipped double Q learning(Fujimoto et al., 2018)), which works better than DDPG with one critic. Parameters are mostly the same as in FetchEnv, except for using 30 parallel environments, 100 steps of unroll per training iteration, 6 million time step replay buffer, 50% hindsight experience, discount $\gamma = 0.99$ and not using observation normalization. Length of the episodes are capped at 100 for Push and 200 for Push and Reach. We train for 14 million frames and evaluate every 200 iterations averaging across 100 episodes.

For all the tasks, the target networks are updated every 40 iterations softly, with weight 0.95 on the existing target network parameters and 0.05 on the incoming. We use Adam optimizer with learning rate $5e^{-4}$ for Atari tasks and $1e^{-3}$ for all others, and $\hat{\epsilon} = 1e^{-7}$ for all tasks.

5.4 RESULTS

We show evaluation performance averaged across three separate training runs. The three training runs used for evaluation use random seeds never used during development.

5.4.1 SAC/DDPG/TD3 VS LB-DR

In Figure 2, we compare lower bounding with discounted return (lb-DR) with SAC, DDPG or TD3 baselines on Atari Breakout and episodic FetchEnv tasks.

[3] We found that the use of observation normalization is critical in reproducing HER results on FetchEnv.

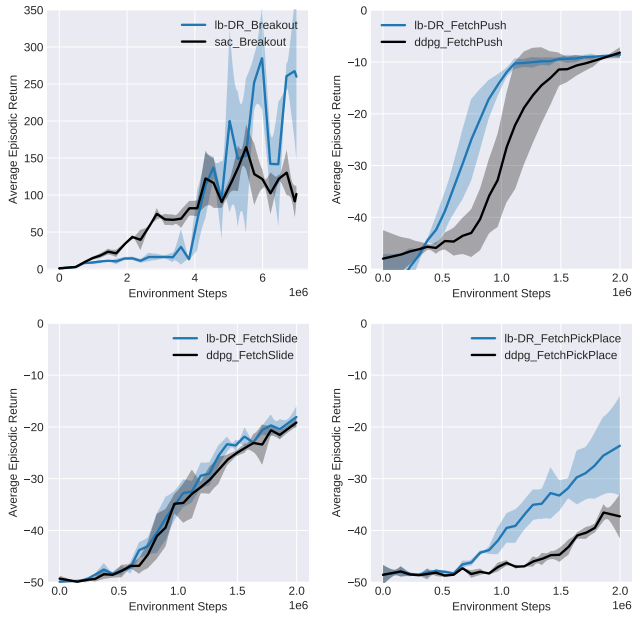

Figure 2: Evaluated average return of value target lower bounding with discounted return (lb-DR) vs SAC or DDPG on Atari Breakout and episodic FetchEnv tasks. Solid curves in the middle are the mean evaluation performance across three seeds, with the shaded areas bounding the minimum and maximum performance of three seeds.

For most environments, lower bounding with episodic discounted return (lb-DR) performs similarly or better than the baselines. For two of the FetchEnv tasks, the improvements in sample efficiency are quite large. For Atari Breakout, the high dimensional image input probably made it harder for the q network to produce the exact action sequences that resulted in the high empirical returns. It learns slower than the SAC baseline initially, but eventually outperforms SAC by a large margin.

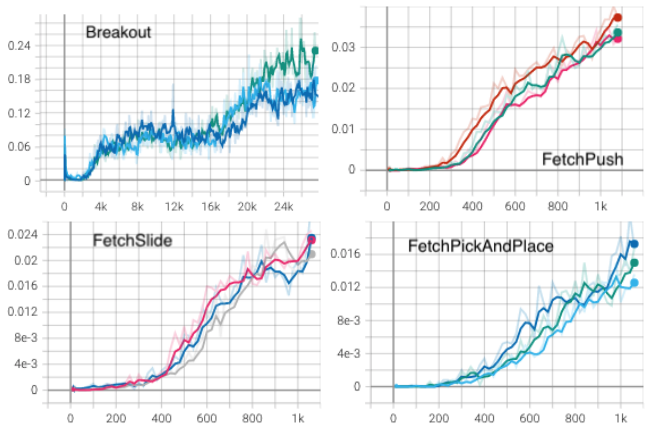

Figure 3: Percentage of training experience where lb-DR return is greater than the Bellman target, on Atari Breakout and episodic FetchEnv tasks, plotted against the number of training iterations. Three seeds are included as three separate curves in the same plot.

In Figure 3, we also look at percentage of experience where lower bounded value target is actually higher than the baseline value target over the course of training. For the FetchEnv tasks where lower bounding improves quite a bit, there seems to be some meaningful percentage of experience where lower bounded value target is above TD return, and the region seems to coincide with where the

average return improves over the baseline. For the Atari Breakout, the new value target improving 10% of the experience doesn't seem to be sufficient, only later, with value target improving over 15% of experience, does lb-DR outperform SAC.

5.4.2 HER VS LB-GD AND LB-DR+GD

In Figure 4, we compare against the much stronger HER baseline on the goal conditioned FetchEnv and Pioneer tasks with our lower bounding with goal distance return (lb-GD) and the lb-DR+GD variant.

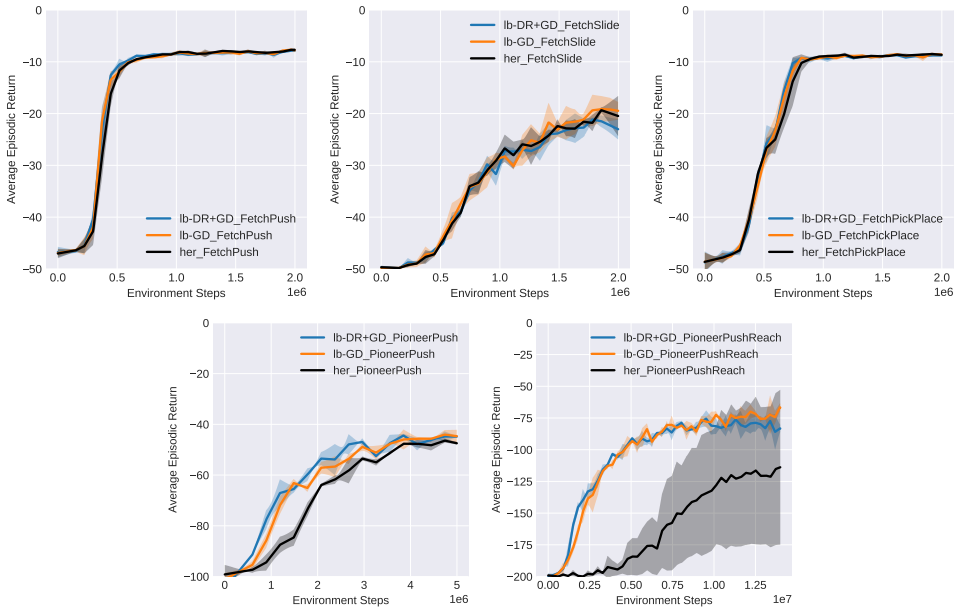

Figure 4: Value target lower bounding with goal distance return (lb-GD) and lb-DR+GD vs TD3 on episodic FetchEnv and Pioneer tasks. Solid curves in the middle are the mean evaluation performance across three seeds, with the shaded areas bounding the minimum and maximum performance of three seeds.

It seems on the easier FetchEnv tasks, lower bounding isn't able to outperform HER, but on the more challenging Pioneer Push and Reach tasks, lower bounding is able to outperform baselines by a large margin. It seems the more complex the task, the wider the margin of gain.

We also looked at the percentage of experience where the lower bounding goal return is higher than the Bellman target (see Appendix A.2). It quickly grows to 1-8% and then slowly drops, matching the region where the new method outperforms the baselines in average return.

5.4.3 PSEUDO-DETERMINISTIC TASKS

When the assumption of a deterministic environment is violated, we had to down weight the lower bounded value target significantly, and it seems there could be small sample efficiency gains during training as well, e.g. Figure 5 for Space Invaders between 2 million to 4 million environment steps. The gains are not as large, and future work is needed to more reliably lower bound value targets under pseudo- or non-deterministic environments.

6 RELATED WORK

This work is most similar to the prior works on lower bound Q learning or Self Imitation Learning (SIL) (Oh et al., 2018; Tang, 2020). We all use empirical return estimated from experience as a value lower bound, and use the lower bound to influence value learning. SIL added an off policy value

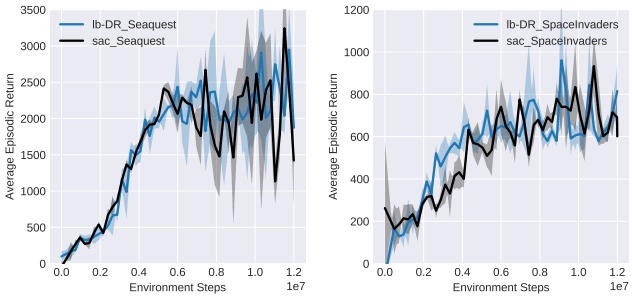

Figure 5: Evaluated average return of value target lower bounding with discounted return (lb-DR) vs SAC or DDPG on Atari Seaquest and Space Invaders. Solid curves in the middle are the mean evaluation performance across three seeds, with the shaded areas bounding the minimum and maximum performance of three seeds.

loss during on-policy (AC or PPO) training, using the empirical return R to directly lower bound the value function ($L^{sil}_{value} = 1/2|v(s) - \max(v(s), R)|^2$). When the value function overestimates, the SIL value loss becomes zero, so it cannot correct value overestimation. We suspect mixing the SIL loss with the loss from the baseline RL algorithms helps bring such overestimations back, but no theoretical guarantee was given. In evaluation, SIL was often compared against on-policy Actor Critic or PPO baselines, so it was not clear how much of the gain was due to lower bounding and how much due to off-policy value learning. In this work, we bound the Bellman value target (Equation 5), so overestimates are automatically corrected via Bellman updates, and convergence is guaranteed. We simply use off-policy algorithms as baselines for a cleaner comparison.

Interestingly, it seems common practice to lower and upper bound the returns to the possible region. For example, Andrychowicz et al. (2017) clipped the value targets to the possible region of $[-1/(1-\gamma), 0]$, which bounds the value targets with two constants.

7 Conclusions

In theory, value target lower bounding converges to the same optimal solution as the original Bellman value iteration.

In practice, several ways of finding value lower bounds using empirical discounted return for episodic tasks are proposed.

Precomputing discounted future return and storing into the replay buffer allows efficient training of value target lower bounding, and can achieve much higher sample efficiency than baselines such as SAC, DDPG or TD3 in most tasks. In tasks with state based input, no mixing weight is needed, whereas in Atari games with image input, tuning the mixing weight allows it to gain slightly faster than the SAC baseline does.

Simple goal distance based return, requiring little extra space or time, can achieve large gains in certain long horizon tasks over HER, and performs similarly as HER in simpler tasks that we experimented with.

7.1 Future Work

There are probably better ways of finding value lower bounds that speed up training even more. There may be ways of using bootstrapped value in computing the lower bound, for n-step return targets or for non-episodic tasks.

Other ways of bounding the value target, e.g. upper bounding, may be worth investigating.

Extending to pseudo- or non-deterministic environments may be possible with a learned reward functions to help average out the randomness in the value bounds, or maybe with distributional RL to properly estimate the value lower bound.

REPRODUCIBILITY STATEMENT

All the experiments are done in simulated environments. We can reproduce the training curves exactly using the same random seed. (We've done that for several of our runs.) Our code change is based on a publicly available RL library, with strong baselines already implemented. Our relatively small code change is committed to a private github repository, which we plan to open source upon publication. When running experiments, the snapshot of the code used to run each experiment is stored together with the results of that experiment, so that we know exactly what is run. Experiment parameters are gin-configured and controlled by our automation script, with each experiment label corresponding to the set of configurations used for that experiment, so there is little room for manual error when running many experiments across different tasks and methods.

In a few cases, we've also run our code on different machines with different GPU hardware and different versions of software, and the results are reproducible up to every float number.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Eric V. Denardo. Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, 9(2):165–177, 1967. ISSN 00361445. URL http://www.jstor.org/stable/2027440.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, 2018. URL http://arxiv.org/abs/1802.09477.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

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*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. *CoRR*, abs/1806.05635, 2018. URL http://arxiv.org/abs/1806.05635.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018. URL http://arxiv.org/abs/1802.09464.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Yunhao Tang. Self-imitation learning via generalized lower bound q-learning. *CoRR*, abs/2006.07442, 2020. URL https://arxiv.org/abs/2006.07442.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

A APPENDIX

A.1 PROOF

Proof. We want to prove that under the new operator, the value function converges to the same optimal value function given by the Bellman operator.

Let v^* be the fixed point and optimal value of the original Bellman operator: $v^* := \mathcal{B}^\infty(v)$, v_1 be any value function, and s any state,

$$
\begin{aligned}
&|\mathcal{M}_f \circ \mathcal{B}(v_1)(s) - v^*(s)| \\
=&|\max(\mathcal{B}(v_1)(s), f(s)) - v^*(s)| \\
&\forall s \text{ where } f(s) > \mathcal{B}(v_1)(s): \\
&\text{above} = |f(s) - v^*(s)| = v^*(s) - f(s) < v^*(s) - \mathcal{B}(v_1)(s) = |\mathcal{B}(v_1)(s) - v^*(s)| \\
&\therefore \text{above} < |\mathcal{B}(v_1)(s) - v^*(s)| \\
&\text{(This is because } f \text{ lower bounds } v^*: v^*(s) \geq f(s) > B(v_1)(s).) \\
&\forall s \text{ where } f(s) \leq \mathcal{B}(v_1)(s): \\
&\text{above} = |\mathcal{B}(v_1)(s) - v^*(s)| \\
\leq&|\mathcal{B}(v_1)(s) - v^*(s)| \\
=&|\mathcal{B}(v_1)(s) - \mathcal{B}(v^*)(s)| \\
\leq&\gamma||v_1 - v^*||_\infty
\end{aligned}
$$

The last line above is because the Bellman operator $\mathcal{B}$ contracts at rate γ.

Hence, $||\mathcal{M}_f \circ \mathcal{B}(v_1) - v^*||_\infty = \max_s |\mathcal{M}_f \circ \mathcal{B}(v_1)(s) - v^*(s)| \leq \gamma||v_1 - v^*||_\infty$.

Therefore, $\forall \epsilon > 0, \forall v_1 \neq v^*, \exists N = \log_\gamma \frac{\epsilon}{||v_1 - v^*||_\infty}$ s.t. $\forall n > N, ||(\mathcal{M}_f \circ \mathcal{B})^n(v_1) - v^*||_\infty < \epsilon$, which concludes the proof that any value function v_1 will converge to v^* under lower bounded Bellman operator $\mathcal{M}_f \circ \mathcal{B}$. □

A.2 PLOTS

We look at the percentage of experience where the lower bounding goal return (lb-GD) is higher than the Bellman target. It seems for FetchEnv tasks, where lb-GD seems to perform only slightly (if any) better than HER, the percentage is quite small. Hindsight relabeling is already producing meaningfully accurate value targets. For Pioneer Push and Reach tasks, lb-GD performs much better in average return, and the percentage of experience on which the lower bound value target is higher than the Bellman target is also much larger.

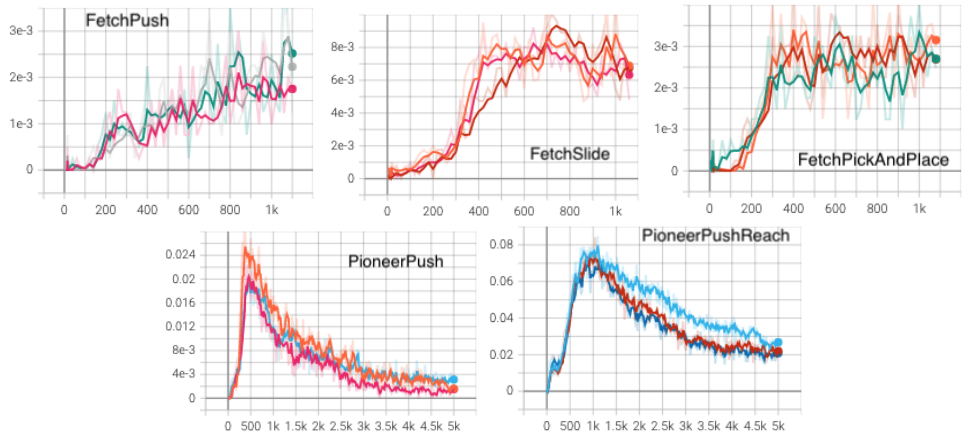

Figure 6: Percentage of training experience where lb-GD return is greater than the Bellman target, on episodic FetchEnv tasks (top row) and Pioneer tasks (bottom row), plotted against the number of training iterations. Three seeds are included as three separate curves in the same plot.

We also provide the learned value during training for lb-DR+GD versus HER. As we can see, value learning accelerates as more experience benefits from the lower bound value targets.

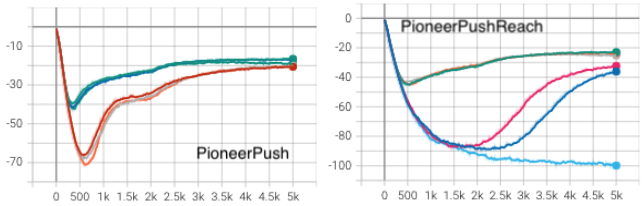

Figure 7: Learned value of lb-DR+GD (top three curves indistinguishably clustered together) vs the value of HER (bottom three curves) for the two Pioneer tasks, evaluated on the training experience and plotted against the number of training iterations. Three seeds are included as three separate curves in the same plot, making a total of six curves in each plot.