DQBS: Deep Q-Learning with Backward SARSA

Jingyu Liu, Yilei Tu, Yunshu Ouyang, Yuyan Zhao
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Deep Q-Network (DQN) with Experience Replay (ER) mechanism is the first algorithm to achieve super-human performance in Atari games and numerous works seek to improve upon them. Most previous works focused on designing architectures or updating rules for the Bellman updates to stabilize training. However, the bias of sampling transitions either randomly or weighted to some priority was often ignored and its validity taken for granted, as they usually perform well in practice. In this work, we designed a simple algorithm called Deep Q-Learning with Backward SARSA (DQBS)[1] which splits a single standard update step of DQN into multiple steps with transitions following chronological backward order. DQBS takes advantage of the Markovian properties of transitions, which assume that the estimated Q-values of state-action pairs can gain more information from those of state-action pairs that follow immediately within trajectories. Each iteration now consists of a normal step which computes the Bellman targets with transitions sampled as usual and several backward steps which calculate the SARSA targets with transitions preceding the previous batch within trajectories. We justified the intuitions behind DQBS with illustrations, conducted ablation studies to prove that each design choice leads to a performance increase, and showed that DQBS outperforms DQN with ER in several Gym environments.**

I. Introduction

Deep Q-Network (DQN) [1] has been shown to exhibit super-human performance in many scenarios. Despite its success, there are numerous works that try to 1) alleviate the overestimation error due to the maximum operator in the bootstrapping Bellman targets [2] and 2) reuse past transitions to increase the sample efficiency [3]. Most variants of DQN are equipped with Experience Replay (ER) [4] buffers which feed the neural networks with random batches to decouple the dependencies of transitions for Bellman updates and to utilize common supervised learning algorithms. But the natural question is whether there is any better choice for the sampling method. In this work, we propose *Deep Q-Learning with Backward SARSA* (DQBS) which 1) applies updates to the Q-function network by the standard Q-learning targets computed by Bellman optimality operator [5] with random transitions from the ER and 2) by SARSA with transitions sampled chronologically backward within trajectories. DQBS stabilizes training and increases convergence rate by allowing newly updated, more accurate estimates of the Q-functions to flow backwards, in a more efficient order than its random counterpart. We tested the algorithm against DQN (with target networks) and DQN with prioritized ER in Gym environments. The results indicate that DQBS exhibits higher stability, lower variance, and better performance in terms of discounted sum of rewards during evaluation. Besides, we also conducted ablation studies to show that our algorithm, despite introducing a few extra hyperparameters, is fairly robust to their values. Our algorithm is simple to implement as it only adds few lines of code on top of DQN variants.

Our main contributions are summarized as follows:

1) We propose a simple algorithm called DQBS which outperforms DQN with random ER and prioritized ER in several Gym environments.
2) We conducted ablation studies to demonstrate the validity and effectiveness of why we chose to sample transitions backward within trajectories and use SARSA instead of Q-Learning targets during the backward updates.

II. Related Works

Our work is centered around two well-known RL algorithms. Q-Learning [6] is an off-policy algorithm which learns to act optimally in Markov Decision Processes with the Bellman Optimality Operator and SARSA is a classic on-policy value evaluation method with provably-fast convergence rate in the tabular case [5]. *Backward Q-learning: The combination of Sarsa algorithm and Q-learning* (BQSA) [7] combines Q-Learning and SARSA to enhance the learning speed and improve final performance. BQSA applies the forward update step by SARSA and the backward update by Bellman targets, which in turn affects the action selection policy. Our motivation differs from theirs as we choose SARSA not for its fast convergence property, but for the fact that it does not contain the maximum operator. Otherwise, the maximum operator in the Bellman targets could potentially prevent the newly updated estimates of the Q-function from flowing backward in the following updates. We will detail this intuition in Section III.

Q-Learning can be combined with neural networks to approximate the Q-function. DQN uses a convolutional neural network trained on raw pixel inputs. It uses

[1] The code is available at https://github.com/Jingyu6/dl_2021

experience replay which randomly samples transitions for better data efficiency and a target network for stability in training.

The experience replay mechanism helps stabilize the update of the Q-function in DQN by breaking the temporal correlations of transitions. One drawback is that uniformly random sampling gives little importance to transitions with large approximation errors. Prioritized experience replay [3] picks more effective transitions by the magnitude proportional to their temporal difference error. Our algorithm can also be easily extended to include the prioritized ER.

III. Proposed Algorithm

In this section, we present a new variant of DQN called DQBS that aims at a faster convergence rate compared to the state-of-the-art algorithms for predicting the Q-function.

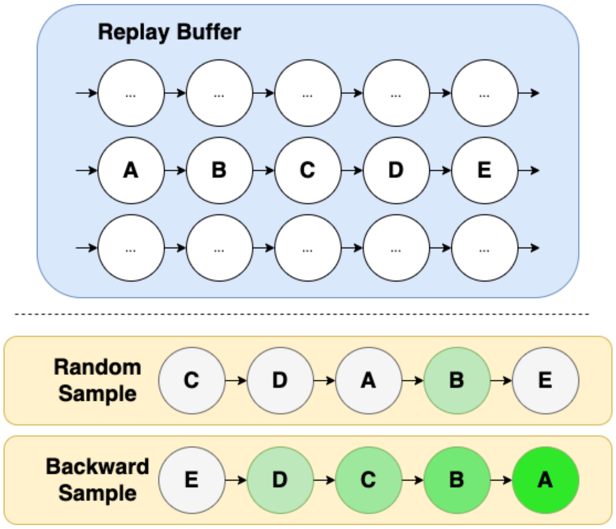


Fig. 1: Comparison between backward sampling and random sampling: the blue box represents the replay buffer containing several trajectories. Each trajectory consists of several states and actions (other information omitted for clarity) that are represented by disks and arrows and ordered chronologically. When we sample state-action pairs, we update their Q-values from left to right in the orange boxes. We assume that the accuracy of the estimate for a state-action pair will improve in this iteration if and only if one of its next state-action pairs gets updated. Green disks denote the state-action pairs whose Q-value estimates gain information, proportional to the color intensity, from the new estimates of their next-state Q-values in this iteration. With random sampling, only the estimate of Q-value for the state-action pair of B will benefit because only B's next state-action pair gets updated before it (note that its approximation will nevertheless be improved by information from the last iteration). With backward sampling, however, all state-action pairs except the first E will have better estimates because their updates follow those of their next states. Therefore, backward sampling in this example will likely produce better results.

a) Intuition of DQBS: One key observation of DQN with ER is that once the Q-value of some state-action is updated, the state-action values that depend on this newly updated value (i.e. those that precede the state-action in the trajectories) will only receive another update when they are sampled later, either randomly or with probabilities proportional to the Bellman errors, which can result in potential inefficiencies. This is illustrated with a toy example in Figure 1. The main idea of our algorithm is to leverage the Markovian properties of dependencies between transitions by modifying the sampling strategy: DQBS propagates backward the newly updated estimates of state-action values so that after each standard Bellman update of the current state-action pairs, the state-action pairs chronologically preceding the current ones are sampled and their values are updated with SARSA. With the backward sampling order, useful updates (i.e. those that actually modify the Q-value) are more likely to happen than DQN, leading to faster convergence.

b) SARSA over Bellman Targets: A possible drawback of using Bellman targets during the backward steps is that the maximum operator can block the information from flowing backward, a phenomenon we call "backward information vanishing". For instance, when we update $Q(s', \tilde{a})$ for some state-action pair and we sample its preceding transition (s, a, r, s'), the new estimate of $Q(s', \tilde{a})$ might not be equal to $argmax_{a'} Q(s', a')$. In this case, the backward steps with the Bellman targets will be useless. To mitigate this problem, we replaced the Bellman targets with SARSA, which can not only make all updates meaningful but also alleviate the overestimation error of DQN due to the value evaluation nature of SARSA. We will investigate the advantage of SARSA over the Bellman targets in the backward steps by experiments in Section IV.

c) Backward Updates: We directly present our algorithm in the function approximation setting. The full algorithm pseudo-code can be found in Algorithm 1. Notice that it is a straightforward generalization from the tabular case. Our algorithm uses experience replay (either random or prioritized) with modified sampling strategy. Each update procedure consists of the following steps:

1) Sample batch transitions B_1 with the default rule (random or prioritized)
2) Optimize the networks with the Bellman targets computed by batch B_1
3) Sample batch transitions B_2 preceding B_1
4) Optimize the networks with the SARSA targets computed by batch B_2

Precisely, after a usual Bellman update of $Q(s, a)$, we do another N "backward updates", where one backward update consists of sampling a transition $(\tilde{s}, \tilde{a}, \tilde{r}, s, a)$, and updating $Q(\tilde{s}, \tilde{a})$ using the SARSA [5] update rule.

d) Backward Sampling for Experience Replay: DQBS requires extending the experience replay buffer so that backward sampling can return a batch of transitions that chronologically precede the input batch transitions. We provide the sampling procedure for the vanilla random ER in Algorithm 2 and prioritized ER in Algorithm 3. Importantly, when using the prioritized ER, we need to update the priority of our transitions both in the Bellman and the SARSA updates. The priority should be proportional to the sum of the losses δ of the sub-trajectory ending at current state-action pairs with length equal to the number of backward steps. The priority update logic which takes in the optimization loss δ and updates the priority map for prioritized ER is summarized in Algorithm 4.

IV. Empirical Experiments

a) Experiment setups: We compared DQBS with DQN, along with some variants of DQBS, BackwardDQN and MultiBatchDQN, to prove the effectiveness of the design choices:

1) BackwardDQN: similar to DQBS with the only difference that the backward update steps follow Q-learning and not SARSA.
2) MultiBatchDQN: consists of sampling multiple batches with the same rule (no backward steps).

Our experiments were carried out using both random ER (RER) and prioritized ER (PER), in three different environments with discrete action space: `CartPole`, `Acrobot` and `MountainCar` from `OpenAI Gym` [8]. Figure 2 compares the performance of each algorithm, with both ER version combination and in every environment. Hyperparameters such as learning rate, ϵ (trade-off value in exploration-exploitation strategy) and batch size were chosen to fit all four algorithms well. All details can be found in the code.

b) Performance analysis: In all but the `Acrobot` with RER, DQBS outperforms the rest in terms of discounted sum of rewards. Notice that DQBS is better in all cases than BackwardDQN and MultiBatchDQN, which indicates that using backward updates and SARSA instead of the Bellman targets can indeed help boost the performance.

c) Ablation studies: We investigated the effects of each of the hyperparameters by ablation studies for DQBS: buffer size, number of backward steps, alpha and beta parameters for the prioritized buffer. The last two are defined in the equation below where for transition i, p_i, $P(i)$, and w_i denote its priority, sampling probability, and importance weight [4]:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \qquad w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{1}$$

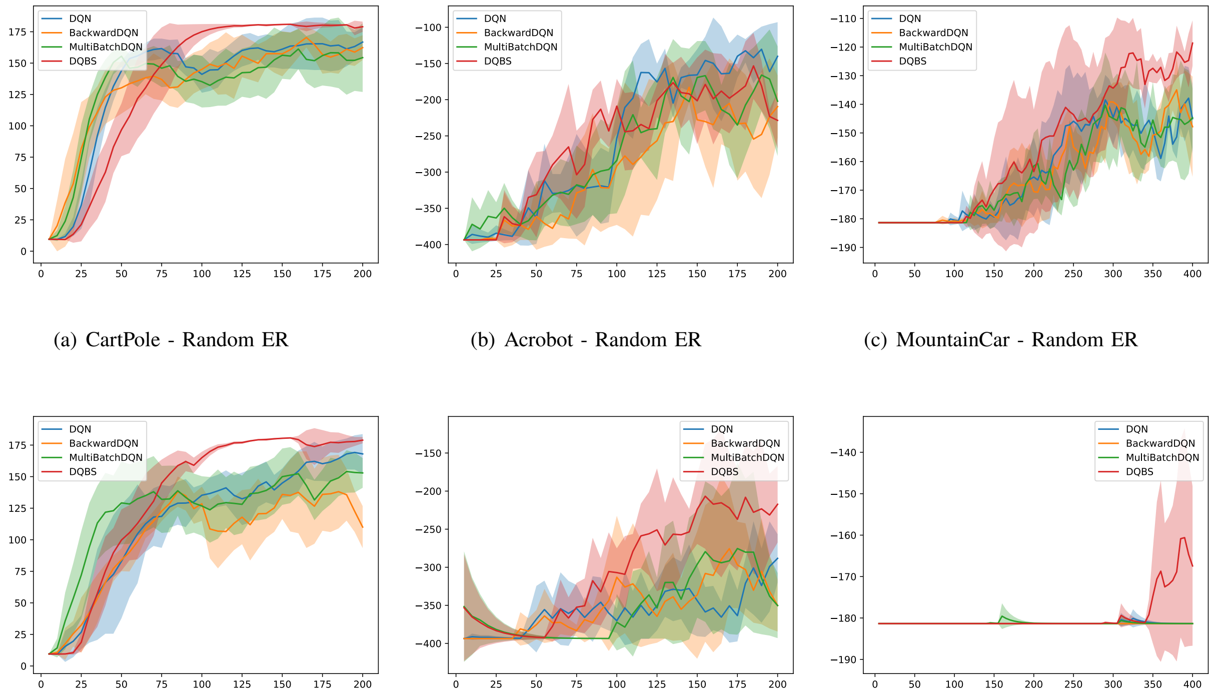
We tested these hyperparameters using DQBS against three environments: `CartPole`, `Acrobot` and `MountainCar` from `Gym`. Below are the observations.

Algorithm 1: Deep Q-Learning with Backward SARSA

```
Environment with state space $\mathcal{S}$ with action
  space $\mathcal{A}$
Initialize $\mathcal{Q}(s, a; \theta)$ with random weights $\theta_0$
Initialize replay buffer $D$ with backward
  sampling method
Observe initial state $s_0$
for each epoch do
    while episode not end do
        $a \leftarrow \epsilon$-greedy of $\mathcal{Q}$
        Take action $a$, observe $r, s'$
        Store transition $(s, a, r, s') \to D$
        Sample batch of transitions $B$ and get
          batch indices $I$ of $B$ from $D$
        Compute the Bellman targets
          $y_{s,a} = r + \gamma \max_{a'} Q(s', a'; \theta)$ for all
          transitions in $B$
        $\delta \leftarrow |y_{s,a} - \mathcal{Q}(s, a; \theta)|$
        $\theta \leftarrow \theta - \alpha \nabla \mathbb{E}_B[\delta^2]$
        if $D$ is prioritized buffer then
            Update priority of $D$ with $I$, $\delta$,
              backward steps
        end
        for number of backward steps do
            Get backward batch transitions $B'$
              and indices $I'$ by backward sampling
              from $D$ with $I$
            Compute the SARSA targets
              $y_{s,a} = r + \gamma Q(s', a'; \theta)$ for all
              transitions in $B'$
            $\delta \leftarrow |y_{s,a} - \mathcal{Q}(s, a; \theta)|$
            $\theta \leftarrow \theta - \beta \nabla \mathbb{E}_{B'}[\delta^2]$
            if $D$ is prioritized buffer then
                Update priority of $D$ with $I$, $\delta$,
                  backward steps
            end
            $I \leftarrow I'$
        end
    end
end
```

Algorithm 2: Random ER Backward Sampling

```
Input: batch size $B$, batch indices $I$
if $I$ is None then
    return random sample of size $B$
end
Compute $I' \leftarrow I - 1$ due to linear buffer
return batch of transitions indexed by $I'$ with
  next state actions indexed by $I$ from the buffer
```



(a) CartPole - Random ER

(b) Acrobot - Random ER

(c) MountainCar - Random ER

(d) CartPole - Prioritized ER

(e) Acrobot - Prioritized ER

(f) MountainCar - Prioritized ER

Fig. 2: Algorithms Comparison: all algorithms are tested against three environments with four runs and fixed random seeds. The horizontal axis denotes the number of epochs and the vertical axis the running average of discounted sum of rewards produced by the greedy policy during evaluation. The mean of rewards over all the runs is plotted with the color lines and the standard deviations are shaded. DQBS outperforms the rest in all except the Acrobot environment with random ER.

Algorithm 3: Prioritized ER Backward Sampling

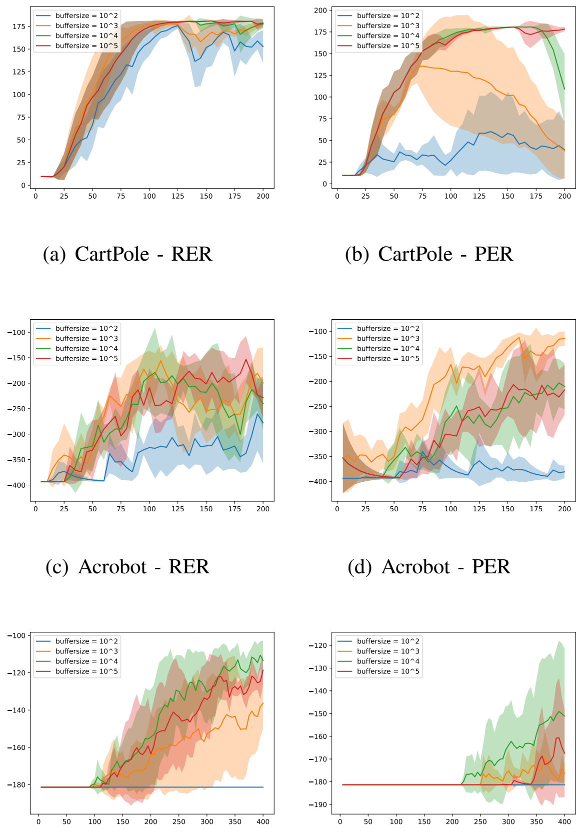
```
Input: batch size $B$, batch indices $I$
if $I$ is None then
    return sample of size $B$ with probabilities
      proportional to priorities, importance
      weights of samples
end
Compute $I'$ based on pointer map storing
  previous transition indices
Calculate new importance weights
return batch of transitions indexed by $I'$ with
  next state actions indexed by $I$ from the buffer,
  importance weights of samples
```

Algorithm 4: Prioritized ER Priority Update

```
Input: batch indices $I$, delta dict $D$, priority dict
       $P$, new delta $\delta$, backward steps $s$
$D[I] \leftarrow \delta$
$P[I] \leftarrow \delta$
define current batch indices $I_{cur} \leftarrow I$
for $s$ steps do
    Find preceding transition indices $I'$ of $I_{cur}$
      based on pointer map
    $P[I] \leftarrow P[I] + D[I']$
    $I_{cur} \leftarrow I'$
end
```

1) **Buffer size** (Fig. 3).
For random ER, our algorithm is not very sensitive to its buffer size. It reaches an optimal performance starting from a certain threshold (around 10^5 for all three environments), then stops progressing (Fig. 3(a), 3(c) and 3(e)). We can conclude that we obtain the best results by choosing any size larger than this threshold, although training time also increases as the buffer takes longer to fill.

When using prioritized ER, we found that very large buffer size does not necessarily yield the best results. We can observe this in Fig. 3(d) and Fig. 3(f). The algorithm continues to improve until 10^3 for `acrobot` and to 10^4 for `mountaincar` but gradually fails as the buffer size continues to increase. Previously unanticipated, we leave the analysis to future work.



(a) CartPole - RER

(b) CartPole - PER

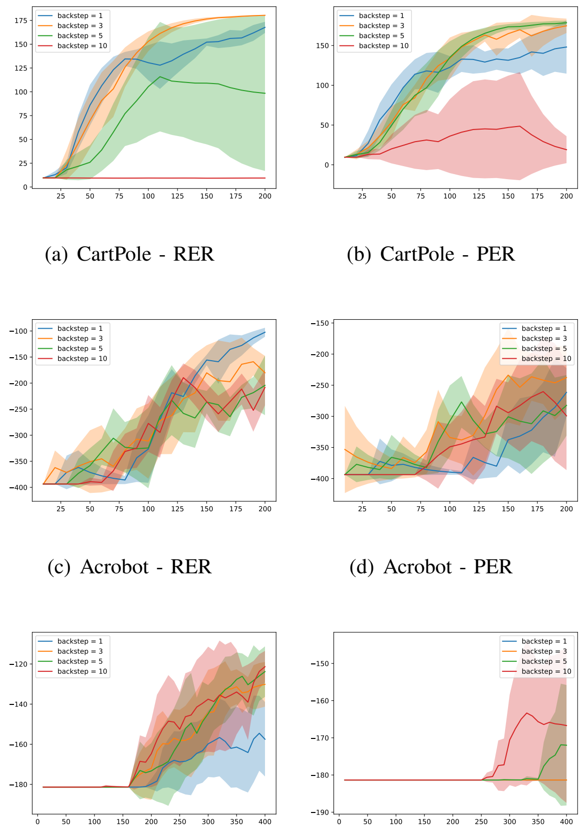
(c) Acrobot - RER

(d) Acrobot - PER

(e) MountainCar - RER

(f) MountainCar - PER

Fig. 3: Buffer Size: different values of buffer sizes are used in DQBS with the same setting as Figure 2



(a) CartPole - RER

(b) CartPole - PER

(c) Acrobot - RER

(d) Acrobot - PER

(e) MountainCar - RER

(f) MountainCar - PER

Fig. 4: Backward Steps: different values of backward steps are used in DQBS with the same setting as Figure 2

2) **Backward steps** (Fig. 4).
   DQBS degenerates to DQN when the backward step size is one and approximates to SARSA when it approaches infinity. As shown in the Fig. 4(a), 4(b), 4(c) and 4(d), $backward\_steps \in [3, 5]$ produced the best empirical results, indicating a fair trade-off between DQN and SARSA. The performance when choosing a larger backward step only improves for the `MountainCar` environment with very little difference compared to using $backward\_steps = 5$ (Fig. 4(e), 4(f)).
3) **Alpha and Beta** The alpha and beta parameters for the prioritized buffer seem to only affect the `MountainCar` environment with $\alpha = 0.2$ and $\beta = 0.01$ being the best values. The small values suggest that prioritized buffer does not work well for this particular environment when combined with DQBS.

V. Conclusions

Inspired by the underlying assumptions of MDPs, we designed a simple algorithm called DQBS which samples transitions following the chronologically backward order within trajectories and applies updates with the Bellman targets and SARSA targets. As shown in the experiments, DQBS allows newly updated estimates of the Q-function to propagate faster, achieving a higher convergence rate and lower variance. DQBS can be applied to any DQN variants and therefore be used with any Actor Critic frameworks, which leaves room for future works.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning."

[2] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning."

[3] T. Schaul, J. Quan, I. Antonoglou, D. Silver, and G. Deepmind, "Prioritized experience replay."

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, cite arxiv:1511.05952Comment: Published at ICLR 2016. [Online]. Available: http://arxiv.org/abs/1511.05952

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[7] Y.-H. Wang, T.-H. S. Li, and C.-J. Lin, "Backward q-learning: The combination of sarsa algorithm and q-learning," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2184–2193, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197613001176

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Z. Openai, "Openai gym."