Reinforcement Learning: Team 1 Final Report

Junghun Ju[*], Jae Yeon Kim[*], Jonghyeok Lee[+], and Soheun Yi[*]

[*]Department of Mathematics, Seoul National University
[+]Department of Statistics, Seoul National University

June 22, 2022

Abstract

For the `Chain MDP` task, we consider the first-visit MC and the modified version of the bootstrapped DQN algorithm by carrying out weighted sampling on the agents to promote exploration while exploitation is ongoing by a sole agent. This modified BDQN method shows prominent adaptability, with respect to the number of states, amidst remarkable sample efficiency. For the `Lava` task, we start our approach through tabular methods with planning, namely Dyna-Q and prioritized sweeping. For generalization, we consider various function approximation methods based on DQN to secure versatility against the change of environment. With some tweaks on the well-known methods, modified methods we call 'prioritized sweeping with transition probability estimation' (tabular) and 'DQN with Q-network' (function approximation) yield satisfactory results. The modified BDQN and DQN with Q-network are chosen as our final algorithms for submission for each task, respectively.

Keywords— DQN, Bootstrapped DQN, Weighted posterior sampling, Prioritized sweeping, Neural network.

1 Methods we tried

The boldfaced algorithms are our final choices for submission.

1.1 `Chain MDP`

- First-visit MC method: Policy is derived according to the G-value calculated within the episode.
- **BDQN (Bootstrapped Deep Q-learning) with weighted sampling**

As a deceptive and easy-to-reach local optimum exists, we focus on choosing an algorithm that readily escapes from local optima and explores ardently. In this context, we borrow the bootstrapped deep Q-learning (BDQN) approach suggested in Osband et al. (2016). We build K DQN agent heads, and Q-function is learned for each agent on a bootstrap sample from the replay buffer. In each episode, greedy action is taken for the sampled single agent. However, there are three critical modifications we make for this specific task.

The part we modify from the original algorithm is first, instead of uniform sampling for the DQN agent heads, that we weigh the probability an agent which derives a high average reward in each episode is selected by taking the softmax function on the exponential moving average of rewards. We expect that learning about the global optimum in all agents will be accelerated through this weighted sampling.

Moreover, to prevent the agents from settling on the suboptimal local optima, we reset the agent when it is determined that the agent stays in any local optimum. Specifically, if the change in the exponential moving average of reward is sufficiently small to be less than the set threshold for a certain number of episodes, the Q-network of the agents is initialized to start a new search.

Lastly, in addition to K DQN agents built for exploration, we build a single DQN agent solely for exploitation. This exploit DQN agent learns from the replay buffer, without being masked or reset. Each episode's probability of taking action by choosing the exploit DQN agent flexibly changes. That is, the probability is added by a hyperparameter α if the minimum of the last five rewards by the exploit DQN agent is greater or equal to the maximum of all the rewards so far by the explore DQN agents, and subtracted by α otherwise. This probability is set to be over 0.1.

We implement the Poisson masks, namely $M_t[k] \sim \text{Poi}(1)$, to mimic the standard bootstrapping behavior. The flowchart of the modified BDQN algorithm is presented in Figure 1.

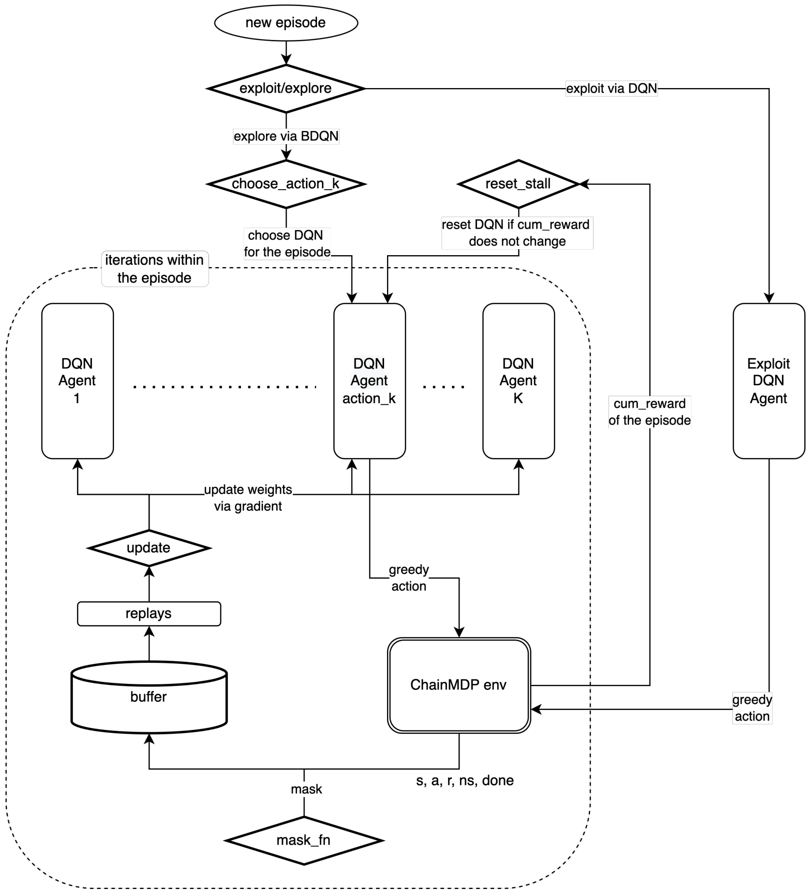


Figure 1: Flowchart of modified BDQN algorithm in `Chain MDP`

1.2 Lava

1.2.1 Tabular methods

Basically we utilize state-space planning methods. With planning, we expect information about the path toward the goal propagates quickly throughout the grid.

- Dyna-Q : We build a model from which to derive simulated experience. For each step in the episode, Q-planning and Q-learning occur in parallel to improve the Q-function. The model is updated for each state-action pair (s, a) to output the most recently experienced (s', r) given (s, a).
- Prioritized sweeping : As in Sutton and Barto (2018), we consider prioritizing the "urgent" ones rather than uniformly sampling a state-action pair. Namely, a queue is maintained according to the value $P = |(\text{target}) - Q(s, a)|$ for each pair. We apply SARSA update instead of Q-update concerning stochasticity in the environment.
- Prioritized sweeping with transition probability estimation : When applying the general `Dyna-Q` method, the difficulty was to adapt to stochasticity. We referred to Manteghi et al. (2015) and made a provisional conclusion that it was because there were many parts close to *lava* in the optimal path. In fact, when stochasticity exists, we observed that the optimal path passes far from *lava* even if the distance gets a little longer, and we thought it would be good to create an agent to learn such a situation. Therefore, we slightly variate the *model* so that it stores $N(s, a, s')$, the number of visits to s, a and s'. By using the *model* of the agent, we attempt to directly estimate the transition probability as $\hat{p}(s', r|s, a) = \frac{N(s,a,s')}{\sum N(s,a,s')}$. Then we estimate the Q-function with full-backup from the formulation $Q(s, a) = \sum \hat{p}(s, a, s')(r + \gamma \max Q(s', a'))$. We provide the pseudocode for this method in Algorithm 1.

Algorithm 1 Prioritized sweeping with transition probability estimation

```
Initialize $Q(s,a)$, $Model(s,a,r,s')$ and $PQueue$ to empty
1) Training
while done do
    $s \leftarrow$ current state, $a \leftarrow \varepsilon$-greedy($s$)
    Execute action $A$, and observe $r, s'$
    if $(s,a,r,s') \in Model$ then
        $N(s,a,r,s') += 1$
    else
        append $(s,a,r,s')$ to $Model$
    end if
    $P = |fullbackup(s,a) - Q(s,a)|$, and insert $(s,a)$ into $PQueue$ with priority $P$
    Planning Steps (Use fullbackup method for update)
end while
2) fullbackup $(s,a)$
for $(S,A,R,S')$ in $Model$ do
    if $(S,A) = (s,a)$ then
        $\hat{p}(s',r|s,a) = \frac{N(s,a,s')}{\sum N(s,a,s')}$, $Q(s,a) = \sum \hat{p}(s,a,s')(r + \gamma \max Q(s',a'))$
    end if
end for
```

1.2.2 Function approximation methods

We carry out several variations of DQN methods. We began with a deep search by a neural network with layers of depth 5. However, this deep method required many episodes to attain optimality and therefore showed low sample efficiency, and in addition, the time required for calculation was too long. No prominent improvement was made by some tweaks in deep search, thus we utilize shallow layers to obtain better performance.

- Function approximation with DNN : With a 2-dimensional feature vector, we utilize Deep NN of depth 5.
- UCB-like algorithm : We found DNN had difficulty escaping around the starting point in the buffer. Therefore, we encourage exploration by giving bonuses on the values of unexperienced state-action pairs.
- **DQN with Q-network** : Even implementation of UCB did not lead to a noticeable improvement, so we tried DQN with shallow layers; as this network is not 'deep', we named this algorithm 'DQN with Q-network'. Figure 2 is the flowchart for this method.

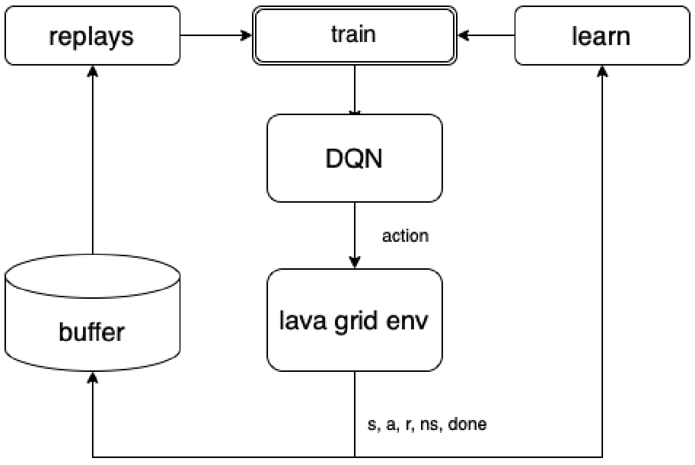


Figure 2: Flowchart of DQN with Q-network in `Lava`

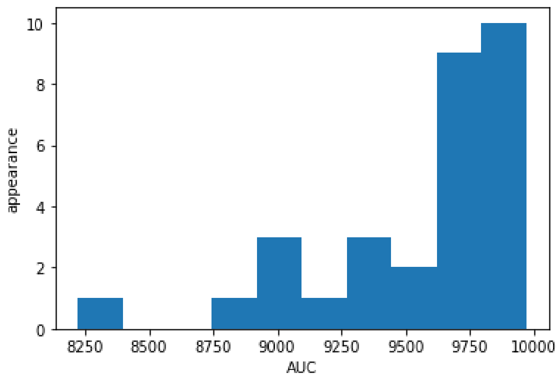
2 Results

Among the algorithms we tried, we list the results for those that showed worthy performance compared to the others. We omit the performance scores as the following algorithms all successfully attain the possible maximum value. We run each algorithm for 30 seeds to measure and display sample efficiency and construct a histogram of AUC on episode-reward trajectories.

2.1 `Chain MDP`

As our modified BDQN yields way better performance than the first-visit MC, we solely show the result of BDQN.

Figure 3 summarizes the sample efficiency of modified BDQN. We run 1000 episodes for each of the 30 seeds.



```
count      30.000000
mean     9545.856633
std       408.327662
min      8222.294000
25%      9369.288250
50%      9727.201500
75%      9803.479500
max      9970.044000
```

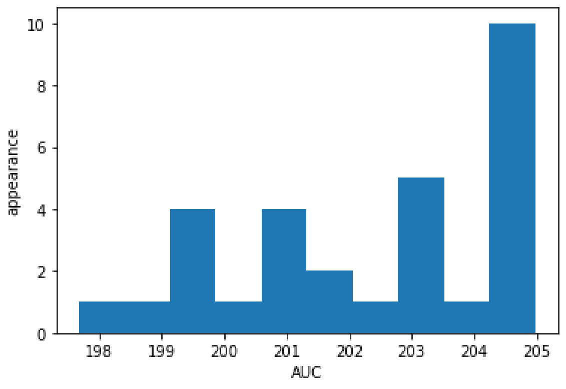
Figure 3: Sample efficiency of modified BDQN. `n_state`=10.

To see how flexible our modified BDQN method is, we reform the environment by adjusting the number of states, which is 10 in the default settings. With `n_state` = 15 and 20, modified BDQN produces satisfactory results. The episode-reward trajectory plots are listed in A.1.

2.2 Lava

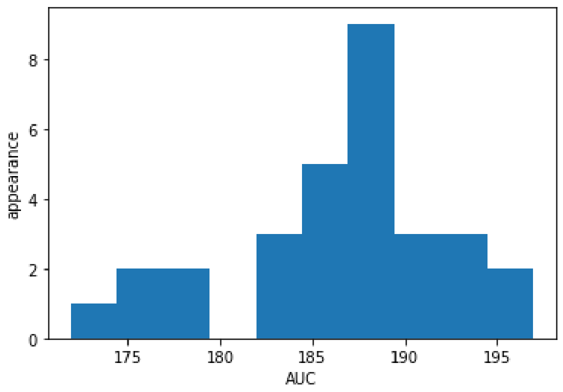
We summarize the algorithms that give the best results, one for each tabular method and the function approximation method, namely prioritized sweeping with transition probability estimation and DQN with Q-network, respectively. To save some time, we run only 300 episodes for sample efficiency estimation for each seed. These short runs are justified, as shown in A.2, for each algorithm tends to converge well way before 300 episodes.

Figure 4 and 5 show the sample efficiency records for prioritized sweeping with transition probability estimation and DQN with Q-network, respectively. Note that, as expected due to the rule of thumb that the tabular methods work better in this setting with abuse of information about the environment, prioritized sweeping yields better results than DQN.



```
count     30.000000
mean     202.386667
std        2.212568
min      197.680000
25%      200.742500
50%      203.045000
75%      204.602500
max      204.970000
```

Figure 4: Sample efficiency of prioritized sweeping with transition probability estimation. Grid shape = 6×10, `stochasticity` = 0.



```
count      30.000000
mean      186.461000
std         6.114923
min       171.910000
25%       184.132500
50%       188.115000
75%       189.800000
max       196.970000
```

Figure 5: Sample efficiency of DQN with Q-network. Grid shape = 6×10, `stochasticity` = 0.

We measure the adaptability of two methods via stochasticity in the dynamics, i.e. the probability of failing to take the intended action, and the change of the grid map size. In particular, we consider `stochasticity` $= 0, 0.02, 0.05$, and 15×15 grid map. Except for the failure of the DQN with the Q-network method in the environment of stochasticity 0.05, they show a good adaptation to the changing environments overall. A.3 delivers the specific results.

3 Conclusion & Suggestions

Due to our evaluation result, we choose the modified BDQN algorithm with weighted sampling for the `Chain MDP` task, and DQN with Q-network for the `Lava` task as our final methods. Speaking of the `Lava` task, although the result is better with the prioritized sweeping type method, we discard this as the tabular method requires the number of states, and it could be considered as the abuse of information about the environment in fully generalized settings.

In the modified BDQN algorithm of the `Chain MDP` task, the most challenging part was to take DQN agents out of the local minima in terms of episodic reward, and we were able to successfully resolve this part by resetting the stall agents. Furthermore, to maximize and stabilize episodic reward, we have added a DQN agent which solely pursues exploitation and adaptively sets the probability of deciding actions via this exploitation DQN agent when each episode starts. This agent achieved the maximum performance(500 for 50 episodes) and arguably high sample efficiency(9545 for 1000 episodes). The adaptive method for deciding exploitation probability seems to have much room for improvement, although we could not achieve those due to the lack of computational power.

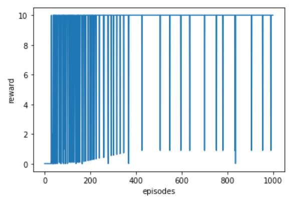
Considering the tabular solutions for the `Lava` task, a prioritized queue promotes the essential updates in the Dyna-Q method, leading to faster learning with less iteration. This improvement by prioritized sweeping is assumed to be generalized and even strengthened in the general environment of larger combinations of tuples in the buffer.

Our biggest challenge was resolving the deep search failure on the `Lava` task. The main focus of this task was to approximate the Q function with excessively high precision since a minuscule error in Q values would render a different greedy policy far from an optimal one. Although we tried to perform the function approximation with a depth 5 network utilizing the full computing power we possess, the physical limit of such capacity hindered our approximation from being precise enough to return an optimal policy. To detour this hardship, we used a shallower network for function approximation and enlarged input dimension from 2 to the dimension of non-modified state feature dimension and consequently achieved a successful performance and sample efficiency(186 for 30 episodes).

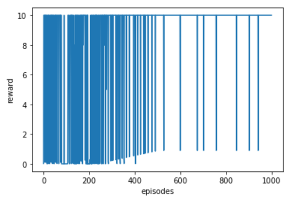
One thing more, our implementation of the UCB-like algorithm was dissatisfactory. For this, considering where UCB operates successfully, it fortunately enables exploration to the unseen region and fosters escape from local optima. However, applying the UCB bonus as a function approximation on the function approximation method also sophisticates the problem. Also, as there exists a negative reward of -0.01 on failing to reach the goal, it seems that exploration itself will not be lacking. Therefore, whereas exploration is an important factor in this task, our UCB-like algorithm is not helpful enough.

A Additional plots

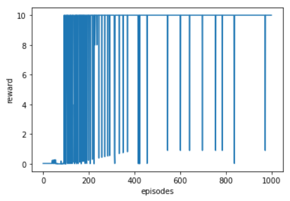
A.1 Modified BDQN : adaptability



(a) `n_state = 10`



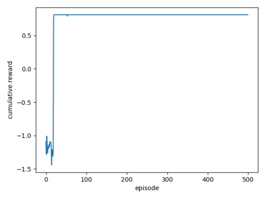
(b) `n_state = 15`



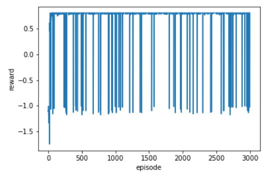
(c) `n_state = 20`

Figure 6: Adaptability of modified BDQN for the number of states

A.2 `Lava` : episode-reward trajectory



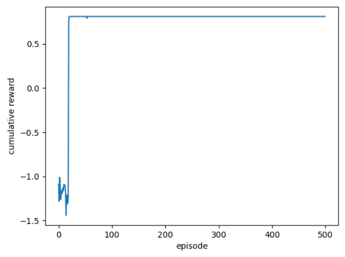
(a) P. sweeping with trans. prob. est.



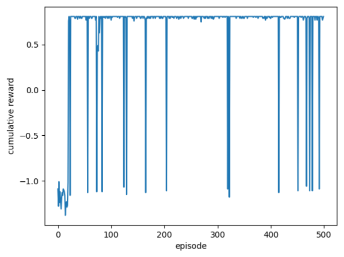
(b) DQN with Q-network

Figure 7: Episode-reward trajectory plots for **`Lava`** solving algorithms

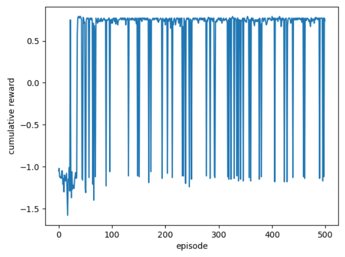
A.3 `Lava` : adaptability



(a) Stochasticity $= 0$



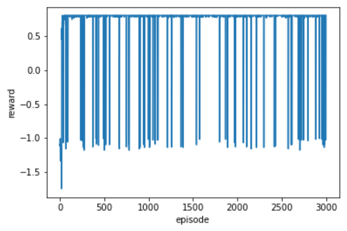
(b) Stochasticity $= 0.02$



(c) Stochasticity $= 0.05$

Figure 8: Adaptability of prioritized sweeping with transition probability estimation w.r.t. stochasticity

In the case of DQN with Q-network with `stochasticity` $= 0.05$, it completely fails.


(a) Stochasticity $= 0$

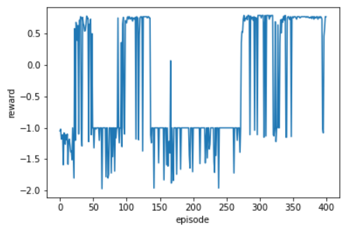
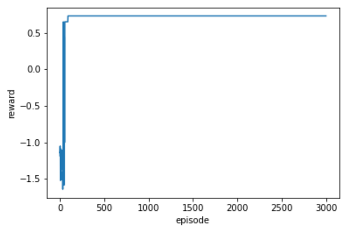

(b) Stochasticity $= 0.02$

Figure 9: Adaptability of DQN with Q-network w.r.t. stochasticity


(a) Prioritized sweeping with transition probability estimation

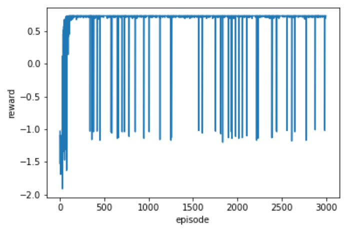
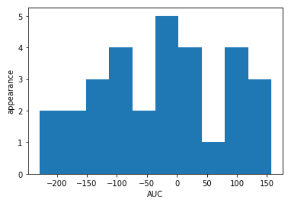
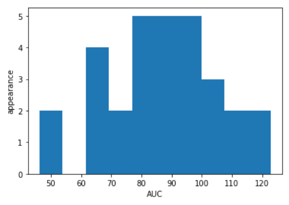
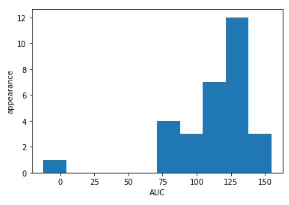

(b) DQN with Q-network

Figure 10: Adaptability of `Lava` solving methods w.r.t. grid size


(a) DQN with Q-net. ; *Stochasticity* $= 0.02$


(b) DQN with Q-net. ; 15×15 grid


(c) P. sweeping with est.; *Stochast.* $= 0.05$

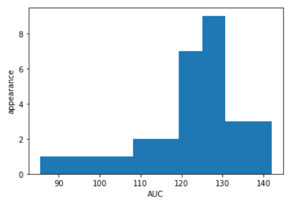

(d) P. sweeping with est. ; 15×15 grid

Figure 11: Comprehensive results for adaptability on the `Lava` task

References

Manteghi, S., Parvin, H., Heidarzadegan, A., and Nemati, Y. (2015), "Multitask Reinforcement Learning in Nondeterministic Environments: Maze Problem Case," in *Mexican Conference on Pattern Recognition*, Springer, pp. 64–73.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016), "Deep exploration via bootstrapped DQN," *Advances in neural information processing systems*, 29.

Sutton, R. S. and Barto, A. G. (2018), *Reinforcement learning: An introduction*, MIT press.