Knowledge Sharing for Reinforcement Learning: Writing a BOOK

Simyung Chang[1], **YoungJoon Yoo**[2], **Jaeseok Choi**[1], **Nojun Kwak**[1]
Seoul National University
[1]{timelighter, jaeseok.choi, nojunk}@snu.ac.kr, [2]yjyoo3312@gmail.com

Abstract

This paper proposes a novel deep reinforcement learning (RL) method integrating the neural-network-based RL and the classical RL based on dynamic programming. In comparison to the conventional deep RL methods, our method enhances the convergence speed and the performance by delving into the following two characteristic features in the training of conventional RL: (1) Having many credible experiences is important in training RL algorithms, (2) Input states can be semantically clustered into a relatively small number of core clusters, and the states belonging to the same cluster tend to share similar Q-values given an action. By following the two observations, we propose a dictionary-type memory that accumulates the Q-value for each cluster of states as well as the corresponding action, in terms of priority. Then, we iteratively update each Q-value in the memory from the Q-value acquired from the network trained by the experiences stored in the memory. We demonstrate the effectiveness of our method through training RL algorithms on widely used game environments from OpenAI.

1 Introduction

Reinforcement learning (RL) has been widely used to study control systems consisting of complex input states and actions, and applied to various research fields [25, 1]. However, it is not easy to approach the RL problem as a direct cost minimization problem because of the constraint that it is difficult to immediately obtain the output according to the input. Therefore, various methods such as Q-learning [3] and policy gradient [27] have been proposed to solve the RL problems.

Recently, a number of neural-network (NN)-based RL methods [18, 28, 17] have been proposed and achieved significantly improved performance in complex environments. These algorithms approximate the dynamic-programming (DP)-based optimal reinforcement learning [12] through the neural network. However, this process has the problem that the Q-values for independent state-action pairs are correlated, which violates the independence assumption. Thus, this process is no longer optimal. [31]. This results in differences in performance and convergence time depending on the experience used to train the network.

To solve the problem, recent deep-learning-based RL algorithms [18, 28, 17, 24, 20] have utilized experience memory in the learning process [15], which stores batches of state-action pairs (experiences) that frequently appear in the network for repetitive use in the future learning process. Also, in [23], a method of prioritized experience memory that finds priorities of each experience is proposed, based on which a batch is created. Eventually, the key to creating such a memory is to compute the priorities of the credible experiences so that learning can focus on the reliable experiences. Therefore, we have to solve the problem of (1) how to acquire the credible experiences and (2) how to configure the priority of the credible experience.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Also, we have observed that in many cases, input states can be grouped into a relatively small number of clusters, and we can safely suppose that the states belonging to the same cluster are likely to share a similar Q-value for a given action. This is a crucial assumption through which the training speed can be greatly accelerated. However, this aspect has not been widely utilized in the RL literatures, yet.

In this paper, we propose a new training method for deep RL that extracts credible experiences dynamically during the network learning process and constructs a dictionary-type memory, which is again utilized in the training of the network afterwards. Different from the existing methods [18, 23] that just utilize the memory as a batch for network training, we update the Q-value corresponding to each pair of state cluster and action in the dictionary using the Q-value computed from the deep network. This process of Q-value update is similar to the DP-based RL method. After this dictionary, termed as 'BOOK', is made, the network parameters are updated by the contents in the BOOK, iteratively. The BOOK that stores the credible experiences is mainly composed of a set of tuples consisting of cluster of states, action, and the corresponding Q-value. By utilizing the BOOK for RL, we achieved meaningful improvements of the performance as well as the convergence speed.

The proposed method provides more efficient network learning strategy than the existing random-batch-based deep RL [18, 28], and it is experimentally shown that the small amount of data stored in the BOOK is sufficient for training the RL network, providing even higher performance than the existing methods. Also, The BOOK can be utilized in a variety of deep RL algorithms because the structure of BOOK is independent of the networks design of the RL.

The contributions of the proposed method are as follows: 1) The dictionary termed as BOOK that stores the credible experience expressed by the tuple (cluster of states, action, and the corresponding Q-value) is proposed. 2) The method for measuring the priority of each experience in the BOOK is proposed. 3) The training method for RL that utilizes the BOOK is proposed, which is inspired by DP and is applicable to diverse RL algorithms.

To show the efficiency of the proposed method, it is applied to the major deep RL methods such DQN [18] and A3C [17]. The qualitative as well as the quantitative performances of the proposed method are validated through the experiments on public environments published by OpenAI Gym[6].

2 Background

The goal of RL is to estimate the sequential actions of an agent that maximize cumulative rewards given a particular environment. In RL, Markov decision process (MDP) is used to model the motion of an agent in the environment. It is defined by the *state* $s_t \in \mathbb{R}^S$, *action* $a_t \in \{a_1, \ldots, a_A\}$ which occurs in the state s_t, and the corresponding *reward* $r_t \in \mathcal{R}$, at time step $t \in \{1, \ldots, T\}$. We term the function that maps the action a_t for a given s_t as the *policy*, and the future state s_{t+1} is defined by the pair of the current state and the action, (s_t, a_t). Then, the overall cost for the entire sequence from the MDP is defined as the accumulated discounted reward, $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, with a discount factor $\gamma \leq 1$.

Therefore, we can solve the RL problem by finding the optimal policy that maximizes the cost R_t. However, it is difficult to apply the conventional optimization methods in finding the optimal policy. It's because we should wait until the agent reaches the terminal state to see the cost R_t resulting from the action of the agent at time t. To solve the problem in a recursive manner, we define the function $Q(s_t, a_t) = \mathbb{E}[R_t | s = s_t, a = a_t, \pi]$ denoting the expected accumulated reward for (s_t, a_t) with a policy π. Then, we can induce the recurrent Bellman equation [3]:

$$Q(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}). \tag{1}$$

It is proven that the value of $Q(s_t, a_t)$ (Q-value) for all time step t satisfying the equation (1) can be calculated by applying dynamic programming (DP), and the resultant Q-values are optimal [12]. However, it is practically impossible to apply the DP method when the number of state is large, or the state is continuous. Recently, the methods such as Deep Q-learning (DQN), Double Deep-Q-learning (DDQN) solve the RL problem with complex state s_t by using approximate DP that trains Q-network. The Q-network is designed so that it calculates the Q-value for each action when a state is given. Then, the Q-network is trained by the temporal difference (TD) [30] method which reduces the gap between Q-values acquired from the Q-network and those from (1).

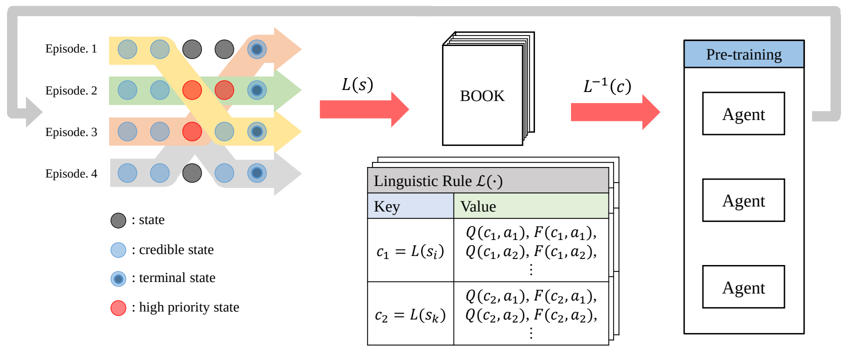


Figure 1: Overall framework of the proposed model. Similar experiences (state-action pairs) from multiple episodes are grouped into a cluster and the credible experiences corresponding to large clusters are written in a BOOK with their Q-values and frequencies Fs. Then, the agent uses this information in training.

3 Related Work

Recently, deep learning methods [18, 11, 28, 29, 17, 23] have improved performance by incorporating neural networks to the classical RL methods such as Q-learning [30], SARSA [22] and policy searching methods [32, 19] which use TD [26]. Mnih *et al.* [18] and Van *et al.* [11, 28] replaced the value function of Q-learning with a neural network by using a TD method. Wang *et al.* [29] proposed an algorithm that shows faster convergence than the method based on Q-learning by applying dueling network method [10]. Furthermore, Mnih *et al.* [17] applied the asynchronous method to Q-learning, SARSA, and Advantage Actor-Critic models.

The convergence and performance of deep-learning-based methods are greatly affected by input data which are used to train an approximated solution [4] of classical RL methods. Mnih *et al.* [18] and Van *et al.* [28] solved the problem by saving experience as batch in the form of *experience replay memory* [16]. In addition, Prioritized Experience Replay [23] achieved higher performance by applying replay memory to recent Q-learning based algorithms by calculating priority based on the importance of experience.

Also, imitation learning [21, 14, 8] which solves problems through expert's experience is one of the main research flows. This method trains a new agent in a supervised manner using state-action pairs obtained from the expert agent and shows faster convergence speed and better performance using experiences of the expert. On the other hand, it is costly to gather experiences from experts.

Our method is totally different in that (1) Compared to imitation learning, the proposed method differs in the aspect that credible data are extracted from the past data in an unsupervised manner. (2) Compared to the prioritized experience replay [23], where memory is simply a batch for Q-network training, the proposed BOOK is totally different in that it is a memory that stores the Q-value for each cluster of states.

4 Proposed Method

In this paper, our algorithm aims to find the credible experience through many experiences and write it into a BOOK, which can share knowledge with other agents. Figure 1 describes the main flow of the proposed algorithm. First, from the RL network (right), we extract the terminated episode of the agents (left). In leftmost figure of Figure 1, the arrows depicts the credible episodes, and the circles in the episode denotes the state of the experiences included in each episodes. we gather the core and credible experiences among experiences from the episodes. Second, The extracted experience are stored into the BOOK memory we defined. The BOOK has semantic clusters of states as key, and stores the value information of the experience related to the semantic clusters. Third, by using the stored contents in the BOOK, we update the RL network again. we iterate the process until the network converges, and then get Book memory that can be directly applied to RL testing. In the section below, the explanation of the BOOK, which stores the credible experiences, and the RL training method using the elements in the BOOK are described in detail.

4.1 Desigining the BOOK Structrue

Given a state $s \in \mathbb{R}^S$ and action $a \in \{a_1, ..., a_A\}$, we define the memory $\mathcal{B}$ termed as 'BOOK' which stores the credible experience in the form appropriate for lookup-table inspired RL. Assuming the semantic correlation among the states, the input state $s_i, i = 1 \ldots N$ can be clustered into the core K clusters $\mathcal{C}_k \in \mathcal{C}, k = 1, \ldots, K$. To reduce the semantic redundancy, the BOOK stores the information related to the cluster $\mathcal{C}_k$, and the corresponding information is updated by the information of the states s_i included in the cluster. It means that the memory space of the BOOK is $\mathcal{O}(AK)$. To map the state s_i to the cluster $\mathcal{C}_k$, we define the mapping function $L : s \to c_k$, where $c_k \in \mathbb{R}^S$ denotes the representative value of the cluster $\mathcal{C}_k$. We term the mapping function $L(\cdot)$, the representative c_k, and the reward of c_k as *linguistic function*, *linguistic state*, and *linguistic reward*, respectively. To cluster the states and define the *linguistic function*, arbitrary clustering methods or quantization can be applied. For simplicity, we adopt the quantization in this paper.

Consequently, the element of a BOOK $b_{k,j} \in \mathcal{B}$ is defined as $b_{k,j} \in \{c_k, Q(c_k, a_j), F(c_k, a_j)\}$, where $Q(c_k, a_j)$ and $F(c_k, a_j)$ denote the Q-value of (c_k, a_j) and the hit frequency of the $b_{k,j}$. Then, the information regarding the input state s_i is stored into $b_k = [b_{k,1}, \ldots, b_{k,A}]$, where $c_k = L(s_i)$. The Q-value $Q(c_k, a_j)$ is iteratively updated by the $Q^t(s_t = s_i, a_t = a_j)$ which denotes the Q-value from the credible experience $\{s_t, a_t, r_t, s_{t+1}\}$.

4.2 Iterative Update of the BOOK using Credible Experiences

To fill the BOOK memory by credible experiences, we first extract the credible experiences from the entire possible experiences. We extract the credible experiences based on the observation that the terminated episode[1] holds valid information to judge whether an agent's action was good or bad. At least in the terminal state, we can evaluate whether the state-action pair performed good or bad by just observing the result of the final action; for example, success or failure. Once we get credible experience from terminal sequences, then we can get the related credible experiences using the equation 2. More specifically, the BOOK is updated using the experience E_t from the terminated episode $\mathcal{E} = \{E_1, ..., E_T\}$ in backward order, i.e., from E_T to E_1, where $E_t = \{s_t, a_t, r_t, s_{t+1}\}$. Consider that for an experience E_t at time t, the current state, current action, and the future state are $s_t = s_i, a_t = a_j$ and $s_{t+1} = s_{i'}$, respectively. Also, assume that $s_i \in \mathcal{C}_k$. Then, the Q-value $Q(c_k, a_j)$ stored in the content $b_{k,j}$ is updated by (2),

$$Q(c_k, a_j) = \beta Q(c_k, a_j) + (1 - \beta) Q^t(s_i, a_j), \tag{2}$$

where

$$Q^t(s_i, a_j) = r_t + \gamma \max_{a'} Q(s_{i'}, a'). \tag{3}$$

$$\beta = F(c_k, a_j) / \{F(c_k, a_j) + F(L(s_{i'}), \arg\max_{a'} Q(s_{i'}, a'))\}. \tag{4}$$

Here, $F(c_k, a_j)$ refers to the hit frequency of the content $b_{k,j}$. The term $Q^t(s_i, a_j)$ denotes the estimated Q-value of (s_i, a_j) acquired from the RL network. In (4), $F(L(s_{i'}), \arg\max_{a'} Q(s_{i'}, a'))$ is initialized to 1 when the term regarding $L(s_{i'})$ is not yet stored in the BOOK. We note that we calculate $Q^t(s_t, a_t)$ from $Q^t(s_{t+1}, a_{t+1})$ in backward manner, because only the terminal experience E_T is fully credible among the episode $\mathcal{E}$ acquired from the RL network. The update rule for the frequency term $F(c_k, a_j)$ in the content $b_{k,j}$ is defined as

$$F(c_k, a_j) = \min(F(c_k, a_j) + F(L(s_{i'}), \arg\max_{a'} Q(s_{i'}, a'), F_l), \tag{5}$$

where F_l is the predefined limit of the frequency $F(\cdot, \cdot)$. The frequency $F(\cdot, \cdot)$ is reduced by 1 for every predefined number of episodes to avoid $F(\cdot, \cdot)$ from being continually increasing. To extract the episode $\mathcal{E}$, we can use arbitrary deep RL algorithm based on Q-network. Algorithm 1 summarizes the procedure of writing a BOOK.

4.3 Priority Based Contents Recoding

In many cases, the number of clusters becomes large, and it is clearly inefficient to store all the contents without considering the priority of a cluster. Hence, we maintain the efficiency of BOOK

[1] An episode denotes sequences of state-action-reward.

Algorithm 1 Writing a BOOK

```
Define linguistic function $L$ for states and reward
Initialize BOOK $\mathcal{B}$ with capacity $N$ and linguistic function $L$
for episode $= 1, \ldots, M$ do
    Initialize Episode memory $\mathcal{E}$
    Get initial state s
    for $t = t_{start}, \ldots, t_{terminal}$ do
        Take action $a_t$ with policy $\pi$
        Receive new state $s_{t+1}$ and reward $r_t$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{E}$
        Perform general reinforcement algorithm
    for $t = t_{terminal}, \ldots, t_{start}$ do
        Take transition $(s_t, a_t, r_t, s_{t+1})$ from $\mathcal{E}$
        $c_k = L(s_t)$
        Update $Q(c_k, a_t)$, $F(c_k, a_t)$ with equation (2), (3), (4), (5)
    if $episode\%T_{decayPeriod} == 0$ then
        Decay $F(c_k, a_j)$ for all $k \in \{1, \ldots, K\}$ and $j \in \{1, \ldots, A\}$
```

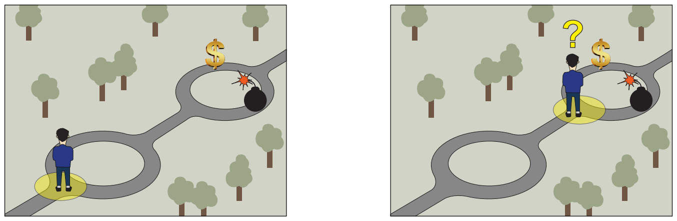


Figure 2: Illustration of the concept of the important term and the linguistic state. The left image shows little difference in expectation of reward according to different choice of action, but in the right image, there is a large difference in the expected value of the reward depending on the action. The yellow circle in each image represents a linguistic state (area) covering the corresponding states (points) sharing the similar environmental semantics.

by continuously removing contents with lower priority from‘the BOOK. In our method, the priority $p_{k,j}$is defined by the product of the frequency term $F(c_k, a_j)$ and the importance term $I(c_k)$,

$$p_{k,j} = I(c_k)F(c_k, a_j). \tag{6}$$

The importance term $I(c_k)$ reflects the maximum gap of reward for choosing an action for a given linguistic state c_k, as the following:

$$I(c_k) = \max_a Q(c_k, a) - \min_a Q(c_k, a). \tag{7}$$

Figure 2 shows the concept of the importance term. At the first crossroad (state) in the left, the penalty of choosing different branches (actions) is not severe. However, at the second crossroad, it is very important to choose a proper action given the state. Obviously, the situation in the right image is much crucial, and the RL should train the situation more carefully.

Now, we can keep the size of the BOOK as we want by eliminating the contents with lower priority $p_{k,j}$. We have shown through experiments that we can obtain good performance even if a relatively small-sized BOOK is used for training. See section 5 for more detailed analysis.

4.4 RL Network Training using the BOOK

As shown in Figure 1, we train the RL network using the BOOK structure that stores the experience from the episode. The BOOK records the information of the representative states that is useful for RL training. The information required to learn the general reinforcement learning algorithm can be obtained in the form of $(s, a, Q(s, a))$ or $(s, a, V(s), A(s, a))$ through our recorded data. Here, $V(s)$ and $A(s, a)$ are the value of the state s and the advantage of the state-action pair (s, a).

To utilize the BOOK in the learning of the environment, the linguistic state c_k has to be converted to the real state s. The state s can be decoded by implementing the inverse function $s = L^{-1}(c_k)$, or one of the state $s \in \mathcal{C}_k$ can be stored in the BOOK as a sample when the BOOK is made.

In the first case of using Q-value $Q(s,a)$ in the training, the recorded information can be used as it is. In the second case, $V(s)$ is calculated as the weighted sum of the $Q(s,a)$ and the difference between the Q-value and the state value V is used as the advantage $A(s,a)$ as follows:

$$V(s) \approx \frac{\sum_{a_i} F(c_k, a_i) Q(c_k, a_i)}{\sum_{a_i} F(c_k, a_i)}, \qquad A(s,a) \approx Q(c_k, a) - V(s). \tag{8}$$

The scaling factor α in equation (9) can be adjusted to $Q(s,a)$ or $V(s)$ to control the influence of the Book in training the RL network.

$$Q(s,a) = \alpha Q(s,a), \qquad V(s) = \alpha V(s) \tag{9}$$

We note that our learning process shares the essential philosophy to the classical DP in that the learning process explores the state-action space based on credible $Q(s,a)$ stored in the BOOK memory and dynamically updates the values in the solution space using the stored information. As verified by the experiments, we confirmed that our methods achieved better performance with much smaller iteration compared to the existing approximated DP [12] based RL algorithms [18, 17].

5 Experiments

To show the effectiveness of the proposed concept of BOOK, we tested our algorithm on 4 problems from 3 domains. These are carpole [2], acrobot [9], Box2D [7] lunar lander, and Q*bert from Atari 2600 games. All the experiments were performed using OpenAI [6].

The purpose of the experiments is to answer the following questions: (1) Can we effectively represent valuable information for RL among the entire state-action space and find important states? If so, can this information be effectively transfered to train the RL network? (2) Can the information generated in this way be utilized to train the network in different architecture? For example, can a BOOK generated by DQN be effectively used to train A3C network?

5.1 Performance Analysis

The graphs in Figure 3 show the effectiveness of the proposed method. In these experiments, we first trained the conventional network of A3C or DQN. The performance of these are colored in blue. During the training of the conventional network, a BOOK is written. Then, we tested the effectiveness of this BOOK with two different scenarios. First, we trained the RL network using only the contents of the BOOK as described in Section 4.4. This normally takes much less time (less than 1 minute in our experiments as will be shown in Table 1) than the training of the conventional network. Then, we tested the performance of 100 random episodes without updating the network. The 'yellow' horizontal line in the graph shows the average score of this setting. Second, one step further to the first scenario, we tested the RL with updating the network using the new episodes. The result is given by the 'red' curve in Figure 3.

For the second scenario, we conducted the experiments with two different settings: (1) training the RL network using the BOOK generated by the same learning method, (2) training the RL network using the BOOK generated by the different learning method. For the first setting, we trained the network and BOOK using A3C [17], while in the second, we generated the BOOK using DQN [18] and trained the network with A3C [17]. The results of these two different settings are the upper and the lower rows of Figure 3, respectively.

As shown in Figure 3, the scores achieved from pre-trained networks using a BOOK were almost similar to the highest scores achieved from conventional methods. Furthermore, additional training on the pre-trained networks was quite effective since they achieved higher scores than conventional methods as training progresses.

5.2 Discussion

To further investigate the characteristics of the proposed method, we conducted the experiments by changing the hyper-parameters.

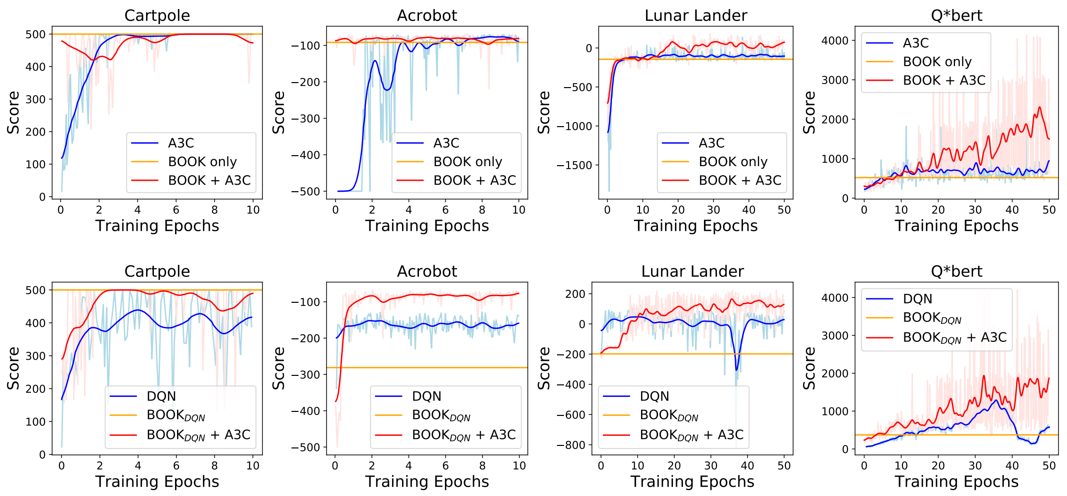


Figure 3: Performance of BOOK based learning. **Blue**: Conventional method (A3C or DQN); **Yellow**: Baseline score averaged on the 100 random episodes. The network was trained only with the BOOK; **Red**: The network was trained using both the BOOK and the new episodes; An epoch corresponds to one hundred thousand transitions (across all threads). Light colors represent the raw scores and dark colors are smoothed scores.

	Cartpole		**Acrobot**		**Lunar Lander**		**Q*bert**	
BOOK Size	**Score**	**Transition**	**Score**	**Transition**	**Score**	**Transition**	**Score**	**Transition**
250	113.98	25.8K	-143.81	330K	-148.70	485K	231.25	78K
500	**500**	**324K**	-158.51	204K	-275.60	215K	371.75	271K
1000	500	324K	**-91.79**	**372K**	**-144.90**	**520K**	436	383K
2000	500	324K	-93.20	363K	-347.71	190K	**520**	**618K**

	Ours	**A3C**	**Ours**	**A3C**	**Ours**	**A3C**	**Ours**	**A3C**
Time	24s	3m 35s	20s	3m 54s	22s	13m 09s	58s	15m 19s

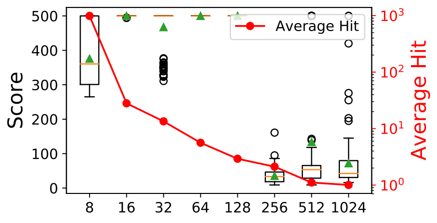
Table 1: The upper table reports the average scores that can be obtained by learning a BOOK of a certain size and the number of transitions that is needed for A3C to get the same score. The table below shows the time required to learn the BOOK and the time required for training A3C to obtain the corresponding highest score written in the bold-faced letter in the upper table.

Learning with different sizes of BOOKs: To investigate the effect of the BOOK size, we tested the performance of the proposed method using Top 250, 500, 1000, 2000 contents in the BOOK. Table 1 shows the score obtained by our baseline network which was trained using only the BOOK in a specified size. Also, in the table, we showed the number of transitions (experiences) that a conventional A3C has to go through to achieve the same score. This result shows that a relatively small number of states can achieve a score similar to that of the conventional network with only the published BOOK. The last row of the table shows the time to learn the BOOK as in Section 4.4 and the time to train A3C to achieve the same score indicated by the bold-faced letter in the upper table. It also indicates that the learning time can be reduced by further training the pre-trained network trained on the BOOK.

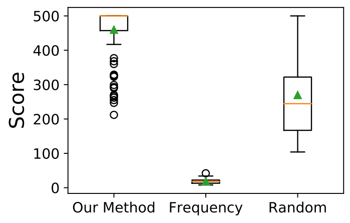
Effects of different quantization levels: In this experiment, we confirmed the performance difference according to the resolutions of linguistic function and the effect of the importance term.

First of all, we differentiated the quantization level and published a BOOK containing 1000 states to check the difference of performance according to the resolutions of linguistic function. Figure 4(a) shows the distribution of scores according to the quantization level and the average number of hits in each linguistic state c_k included in the BOOK.

From Fig. 4(a), we found that the number of hit for each linguistic state decreases exponentially as the quantization level increases. Also, when the quantization level is high, the importance of c_k in equation (7) couldn't be defined and its score decreased because hit ratio becomes low. It can be seen that the highest and stable scores are obtained at quantization level of 64 and 128.



(a) Quantization Level



(b) Priority Methods

Figure 4: (a) Distribution of scores according to the quantization level and the average number of hits in each Linguistic State c_k included in the BOOK. (b) Distribution of scores when we use two different methods than our proposed priority method. All data were tested in Cartpole, and scores were measured in 100 random episodes. The green triangle and the red bar indicates the mean and the median scores, respectively. Blank circles are outliers.

Also, to verify the usefulness of the importance term (7) which is used to determine the priority (6) in our method, we tested the algorithm with different design of the priority; random selection, frequency only, and the proposed priority method. A book capacity N was reduced to 10,000 for this test.

As shown in Figure 4(b), the algorithm applying the proposed priority term achieved the clearly superior performance than other settings. We note that the case only using frequency term marked the lowest performance, even lower than the random case. This is because the learning process proceeds only to the experiences that appear frequently when the priority is set only by the frequency. Correspondingly, the information of the critical, but rarely occurred experiences are not reflected enough to the training and hence, leads to the inferior performance.

5.3 Implementation Detail

We set the maximum capacity N of a BOOK to $100,000$. To maintain the size of the BOOK, only the top 50% experiences are preserved and the remaining experiences are deleted to save new experiences when the capacity exceeds N. As a linguistic rule, each dimension of the input state was quantized into 128 level. We set the discount factor γ for rewards to 0.99. Immediate reward r was clipped from -1 to 1 at Q*bert and generalized with $tanh(r/10)$ for the other 3 environments (Cartpole, Acrobot and Lunar Lander). The frequency limit F_l was set to 20 and the decay period T was set to 100.

Our method adopted the same network architecture with A3C for Atari Q*bert. But for the other 3 environments, we replaced the convolution layers of A3C to one fully connected layer with 64 units followed by ReLU activation. Each environment was randomly initialized. For Q*bert, it skipped a maximum of 30 initial frames for random initialization as in [5]. We used 8 threads to train A3C network and instead of using shared RMSProp, ADAM [13] optimizer was used. All the learning rates used in our experiments were set to 5×10^{-4}. To write a BOOK, we trained only 1 million steps (experiences) for Cartpole and Acrobot and 5 million steps for Lunar Lander and Q*bert. After publishing a BOOK, we pre-trained a randomly initialized network for $10,000$ iterations with batch size 8, using only the contents in the published BOOK. It took up to a minute to learn a BOOK with 1 thread on Nvidia Titan X (Pascal) GPU and 8 CPU cores, for Q*bert, and it took even less time for others experiments as shown in Table 1.

6 Conclusion

In this paper, we have proposed a novel neural network based RL training method inspired by the classical DP. To tackle the problem, we proposed the method that accumulates credible experiences in the BOOK and explores better Q-values in the solution space defined by the state and action, based on the stored information. In doing so, instead of directly writing each state, we clustered similar states by quantization and the corresponding Q-value for each cluster is calculated. Through the experiments, we verified that our method can train the RL network with much higher score and faster convergence speed, compared to the baseline deep RL methods. This paper discovers meaningful progress in deep RL training in that our work introduces a way to extract, manage, share and reproduce the essential information which is critical for efficient RL network training.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007.

[2] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[3] Richard Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

[4] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

[5] Dmitry Bobrenko. Asynchronous deep reinforcement learning from pixels.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[7] E Catto. Box2D: A 2D physics engine for games. 2011.

[8] Kai-Wei Chang, He He, Hal Daumé III, and John Langford. Learning to search for dependencies. *arXiv preprint arXiv:1503.05615*, 2015.

[9] Alborz Geramifard, Christoph Dann, Robert H Klein, William Dabney, and Jonathan P How. Rlpy: a value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, 16:1573–1578, 2015.

[10] Mance E Harmon, Leemon C Baird III, and A Harry Klopf. Advantage updating applied to a differential game. In *Advances in neural information processing systems*, pages 353–360, 1995.

[11] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[12] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Akshay Krishnamurthy, CMU EDU, Hal Daumé III, and UMD EDU. Learning to search better than your teacher. *arXiv preprint arXiv:1502.02206*, 2015.

[15] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[16] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. PhD thesis, Fujitsu Laboratories Ltd, 1993.

[17] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[19] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

[20] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[21] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[22] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[23] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[24] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[25] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[26] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[27] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[28] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[29] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[30] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[31] Paul J Werbos. Approximate dynamic programming for real-time control and neural modeling. *Handbook of intelligent control*, 1992.

[32] Ronald J Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks*, volume 2, pages 601–608, 1987.