2019 IEEE Symposium Series on Computational Intelligence (SSCI)
December 6-9 2019, Xiamen, China

Relevant Experiences in Replay Buffer

Giang Dao and Minwoo Lee
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC 28223
Email: gdao@uncc.edu, minwoo.lee@uncc.edu

***Abstract*—The sampling bias problem creates instability in reinforcement learning. Reusing past experiences with experience replay helps reinforcement learning overcome possible sampling bias. After the success of deep reinforcement learning, other variations of experience replay have been proposed and further improved learning performance. However, how to select samples to store in experience replay methods has not been well investigated. In this paper, we examine if there exists *relevant* (or significant) experiences to be preferably replayed. For systematic discovery of relevant experiences, we adopt the DRL-Monitor and discuss how it improves the efficiency of learning. Comparing with traditional experience replay and prioritized experience replay, we demonstrate improved learning performance in different Atari games.**

***Keywords*—Reinforcement learning; sparse Bayesian reinforcement learning; relevant experiences; replay buffer.**

I. Introduction

Recent advances in deep reinforcement learning have solved a lot of different applications. There are a lot of methods have been built to solve world challenging problems such as robotics [3], allocating cloud computing resource [12], advertisement technology [21], and finance [5]. Reinforcement learning agent, however, learns from trial-and-error in an environment. This includes sequential data sampling that results in a high correlation between two consecutive data, which are affected by the action taken from the previous state. The lack of independence in distributions for training data breaks an assumption of many stochastic gradient descent algorithms. Furthermore, learning from new experience causes forgetting previously learned optimal policies. These were the main issues in traditional reinforcement learning that cause instability [15].

Experience replay [11; 16] stores past experiences in memory and replays some of them by randomly sampling from a uniform distribution. This breaks the direct correlation in training data and simulates independent distribution for gradient descent updates. Simply storing all experiences also prevents possible forgetting problems. Experience replay also improves the data efficiency [20], which suits many different RL algorithms. Deep Q-Network (DQN) [14] has shown the stability of the training process by using experience replay.

After the success of DQN, a fairly large amount of memory seems to be necessary for the experience replay (replay buffer) [14; 10; 1]. Without any justification of the size of the replay buffer, a lot of followup research adopts the heuristically determined size for the memory. However, storing all the experiences and then sampling some from uniform distributions is not the most effective way to use a replay buffer. Thus, it requires investigation on the design of a replay memory: *storing* problem of which experiences to store and *replaying* problem of which one to replay [15].

Recent research focuses on the *replaying* problem to improve the efficiency of experience replay: most notably, Prioritized Experience Replay (PER) [15], Hindsight Experience Replay (HER) [1], and Distributed Prioritized Experience Replay (DPER) [7]. PER modifies the traditional experience replay whereby instead of uniformly choosing experiences from replay buffer, the agent is more likely to sample experiences with higher temporal difference error. HER realizes sample augmentation in order to overcome the rare relevant (reaching goal trajectories) experiences problem with virtual goals and shows much improved performance on learning. For an environment that multiple agents learn asynchronously, DPER adopts PER in distributed learning framework for efficient learning. These approaches make advances on efficient *replaying*, eventual effective learning, but the *storing* problem is not so far addressed.

In this paper, we suggest a simple, but novel method, *Relevant Experience Replay* (RER), as an initial step to solving the *storing* problem. RER adopts DRL-Monitor [4] to identify important experiences to tag them as "relevant." DRL-Monitor, attached to a DRL agent, learns a policy developed by the agent by observing the learning process and collecting important moments to understand the rationale of the agent's decision making. When DRL-Monitor tags some samples as relevant or important, a replay buffer contains both the relevant samples and irrelevant samples. DRL samples from this mixture with a certain ratio and we examine the quality of relevance tagging for learning. This requires a slight modification on the DRL algorithms' sampling module. Our experiments show the efficacy and efficiency of the use of relevant samples identified by DRL-Monitor in 11 different Atari game environments.

Our main contributions in this work include a framework that identifies relevant experiences to improve, which are critical moments of the agent environment interaction, and enforces them to be replayed. This leads to the empirical evaluation of the quality of the proposed relevant experience selection module to tackle the storing problem in a replay buffer in the future.

978-1-7281-2484-1/19/$31.00 ©2019 IEEE

In Section II, we outline the backgrounds for the proposed work. The proposed relevant experience replay is introduced in Section III, and the results of applying our method in different Atari games are presented in Section IV. We draw conclusions in Section V.

II. Background

A. Reinforcement Learning

A reinforcement learning (RL) problem consists of an agent interacting with an environment. The agent perceives a state s_t from the environment and performs an action a_t to the environment. The environment returns the next state s_{t+1} as well as a corresponding reward of the state and action pair r_t.

Through the learning process, the agent provides a policy π that maps states to actions such that: $\pi : \mathcal{S} \rightarrow \mathcal{A}$. At time step t, action a_t is taken based on the current policy π and state s_t: $a_t = \pi(s_t)$. The agent receives reward r_t and perceives s_{t+1} to continue the learning process.

The return is computed with the sum of discounted future rewards $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$ where $\gamma \in [0, 1]$ and T is the timestep that the environment terminates. An action value ($\mathbf{Q}$-value) function is used for estimating the expected given state and action pair return: $\mathbf{Q}(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$. The agent learns by trying to maximize the expected returns using the Bellman equation:

$$\mathbf{Q}^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \epsilon} \Big[r(s_t, a_t) + \gamma \mathbf{Q}^*(s_{t+1}, a_{t+1}) \Big]$$

where $\mathbf{Q}^*$ denotes the optimal $\mathbf{Q}$-value function.

To balance the exploration and exploitation, a near-greedy method, ϵ-greedy, is developed. The method explores the space by uniformly sampling an action from $\mathcal{A}$ with a probability of ϵ, and independently take a greedy action (highest $\mathbf{Q}$-value estimation) with a probability of $1 - \epsilon$.

B. Deep Q-Network

Deep Q-Network (DQN) is one of the deep reinforcement learning (DRL) algorithms, which learns from interaction between an agent and an environment. DQN uses convolutional neural networks to map raw images into $\mathbf{Q}$-values for a set of discrete actions.

At each time step, from the interaction, a state experience tuple (s_i, a_i, r_i, s_{i+1}) is stored into an experience replay buffer. Randomly sampled training data from the replay buffer provide an identical and independent distributions for training the DQN. Thus, it helps avoid possible sampling bias to improve learning performance.

To improve the stability of learning, when approximating $\mathbf{Q}^*$-values, DQN uses two deep neural networks, online network $\mathbf{Q}^{online}(s_t, a_t)$ and target network $\mathbf{Q}^{target}(s_t, a_t)$. The online network is trained using mini-batch gradient descent where the target:

$$y_t = r_t + \gamma \max_{a \in \mathcal{A}} \mathbf{Q}^{target}(s_{t+1}, a).$$

The mean squared loss can be set up as:

$$\mathcal{L} = \Big(y_t - \mathbf{Q}^{online}(s_t, a_t) \Big)^2.$$

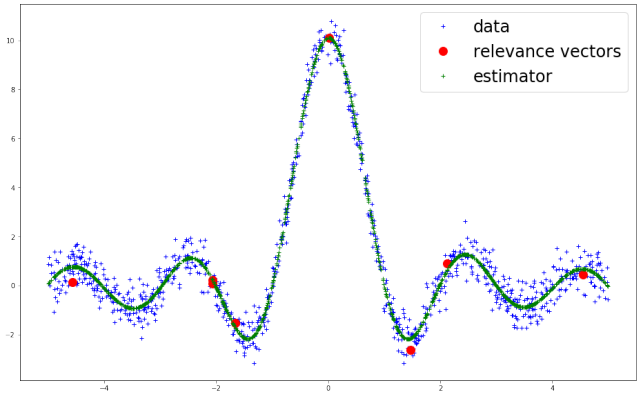


Fig. 1: An example function $f(x) = 10sinc(x) + \mathcal{N}(0, 0.5)$ of applying the relevant vector machine [17] to indicate relevant samples where the blue dots are the data and red dots are relevant samples after trained. Only using the red dots with learned parameters, the method can approximate the true function (green dots) of the noisy data.

The target neural network is identically structured as the online network. The weights of $\mathbf{Q}^{target}$ are periodically update with the weights of $\mathbf{Q}^{online}$. Thus, the $\mathbf{Q}^{target}$ changes the estimation of Q values slower than the $\mathbf{Q}^{online}$. [14] empirically showed the improved stability and performance of this structure in Atari games.

Double Deep Q-Network (Double DQN) [19] extends the DQN. Double DQN computes the target value y_t by using the both online and target networks as follows:

$$y_t = r_t + \gamma \mathbf{Q}^{target} \Big(s_{t+1}, \arg\max_{a \in \mathcal{A}} \mathbf{Q}^{online}(s_{t+1}, a) \Big)$$

Target values are estimated by target network with greedy action from online network. The authors observed further improved performance with Double DQN in their experiments.

C. DRL-Monitor

We shows an example of the sparse Bayesian learning method in Fig. 1. The method is capable of extracting important samples as basis and serves to reconstruct the original function. Sparse Bayesian reinforcement learning (SBRL) [9; 13] is a development of the method to solve complex reinforcement learning problems.

Combining DRL with SBRL, DRL-Monitor [4] records significant experiences during training. Using the recorded experiences, DRL-Monitor explains what an RL agent learns, how the agent builds up it knowledge, and how it can be related to new environment for exploitation.

DRL-Monitor adopts a kernel-based method, sparse Bayesian learning [17]. The monitor re-approximates DRL to extract meaningful experiences for explaining the learning processes and the learned policies. When approximating DRL, the monitor assumes the target $\mathbf{Q}$-values is a weighted sum of the feature vectors with some noise $\boldsymbol{\epsilon}$ such that:

$$\mathbf{Q} = \boldsymbol{\Phi} \mathbf{w} + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon}$ is a zero-mean Gaussian noise with variance σ^2 suggested by [18] as:

$$\sigma^2 = 0.1 \times \text{var}(\mathbf{Q}).$$

$\boldsymbol{\alpha}$, a set of hyper-parameters controlling the strength of the prior over the corresponding weights, is set to be infinity except for one starting

$$\alpha_i = \frac{||\phi_i||^2}{||\phi_i^\top \mathbf{Q}||^2/||\phi_i||^2 - \sigma^2}.$$

The mean and standard deviation of the weights distribution with are computed as:

$$\boldsymbol{\Sigma} = (\boldsymbol{\alpha}\mathbf{I} + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1} \quad \text{and} \quad \boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top\mathbf{Q}.$$

$\boldsymbol{\alpha}$, σ^2, $\boldsymbol{\Sigma}$, and $\boldsymbol{\mu}$ is iteratively re-computed based on *sparsity* and *quality* factors. The algorithm runs until it reaches a certain iteration or a convergent condition is met. The relevant experience is retrieved by tracing the bases left in the model.

III. Relevant Experience Replay

The suggested model leverages DRL-Monitor to systematically select important experiences to tag as relevant experiences for replay. We replay samples from the mixture of the tagged "relevant" experiences with other irrelevant experiences for efficient training as shown in Fig. 2.

Algorithm 1 Relevant Experience Replay (RER)

Input: batch size b, training period K_{train}, monitoring period $K_{monitor}$, total step T, relevant replay ratio $c \in [0, 1]$.
Initialize a replay buffer memory M
Observe an initial state s_0
for $t = 0$ **to** $T - 1$ **do**
  Get action a_t from s_t through policy $\pi(s_t)$
  Execute a_t and collect reward r_t
  Initialize $f_t = 0$ and check terminal d_t
  Store experience $e_t = (s_t, a_t, r_t, d_t, f_t)$ in M
  if $t \mod K_{monitor} = 0$ **then**
    Retrieve $K_{monitor}$ previous experiences from M
    Extract $\mathbf{p}$, $\mathbf{a}$, and $\mathbf{Q}$-values
    Normalize $\mathbf{p}$ using Eq.(2) and map $\mathbf{a}$
    Monitor the DRL agent
    Modify tagging flag f with Eq.(1)
  end if
  if $t \mod K_{train} = 0$ **then**
    Empty the batch memory B
    $R \leftarrow$ relevant experiences with $f_t = 1$
    $I \leftarrow$ irrelevant experiences with $f_t = 0$
    Sample $c \times b$ of $e_t^{(r)}$ from R and add to B
    Sample $(1 - c) \times b$ of $e_t^{(i)}$ from I and add to B
    Perform one step gradient update with B
  end if
end for

A. Training DRL-Monitor

DRL-Monitor answers to the question of what to store during replaying experiences. DRL-Monitor observes a DRL agent's learning progress in a training environment and its application of a learned policy in test environments. From training, the agent tunes the neural network weights to produce Q values for appropriate actions to take. Encoded state representation is developed through multiple neural network layers (i.e. convolutional and fully connected layers). We refer this encoded state vector from the fully connected layer before the output layer as *perception*. Observation of DRL enables DRL-Monitor to recognize important samples from remembered frame images, actions, and rewards.

When the monitor considers all data samples as relevant experiences, the model will be identical to traditional experience replay. This can be avoided by discarding similar samples. To prevent memorizing any possible similar experience, DRL-Monitor measures the similarity by using kernel function. For simplicity, our initial model defines a product kernel of Radial Basis Function (RBF) kernels to measure the similarity of both perceptions and mapped actions as:

$$k\big((\mathbf{p}, \mathbf{a}), (\mathbf{p}_i, \mathbf{a}_i)\big) = k_s(\mathbf{p}, \mathbf{p}_i) \times k_a(\mathbf{a}, \mathbf{a}_i).$$

The product kernel is also used for training DRL-Monitor to construct sparse bases to approximate the DRL $\mathbf{Q}$ function space [4]. The bases built in the monitor are the key data samples capable of estimating any representation that a DRL agent can make. Thus, we treat the kernel bases as the relevant experience to replay for efficient learning.

B. Tagging Relevant Experiences

We assume the data in the replay buffer is sequential presentation similar to taking state after state in the simulation. A single experience sample e_t in a replay memory can be represented as $e_t = (s_t, a_t, r_t, d_t, f_t)$ where the boolean terminal state indicator d_t tells whether the state s_t is terminal or not, and a boolean tagging flag f_t tells the relevance. Then, the experience replay buffer $M = [e_1, \ldots, e_{N_m}]$ where N_m is the number of stored experiences.

f_t is initialized as zero for every new experience comes into the replay buffer. After finishing DRL-Monitor training, $\boldsymbol{\alpha}$, the set of hyper-parameters controlling the strength of the prior over the corresponding weights, tells the significance samples among the constructed bases. Corresponding f_t's are marked as 1:

$$f_t = \begin{cases} 1 & \text{if } \alpha_t < \infty \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

With this implementation, all relevant experiences are tagged with value $f_t = 1$, and other irrelevant experiences are tagged with value $f_t = 0$ in the replay buffer.

C. Replaying Experiences

For informed random sampling with relevance tagging, we sample a training batch which concatenates a portion from a relevant experience set with another portion from the other irrelevant experience set. A batch size b and a ratio $c \in [0, 1]$

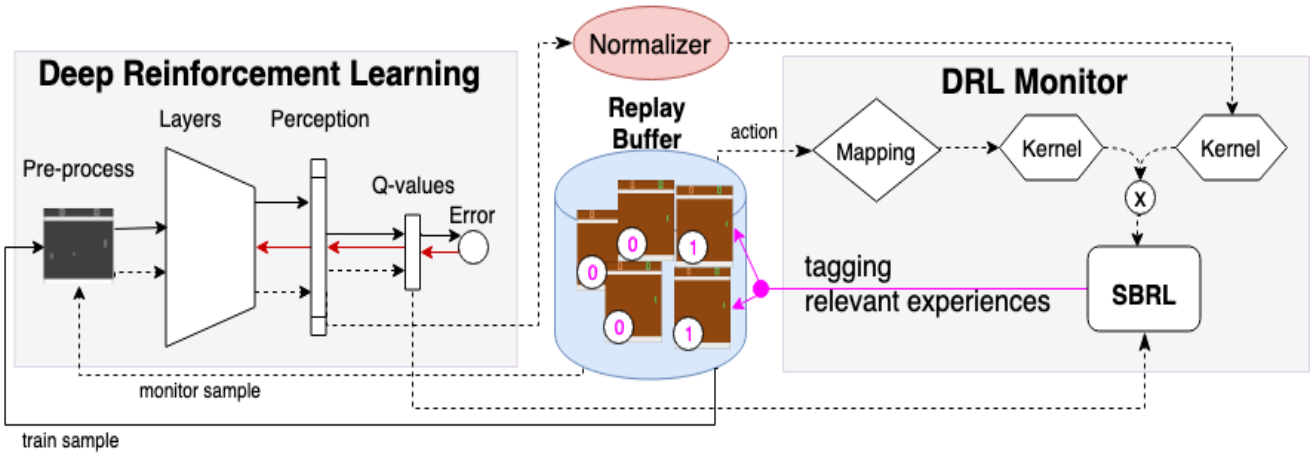


Fig. 2: Relevant experience replay framework with DRL-Monitor. DRL-Monitor identifies and tags relevant experiences in a replay buffer. With a sampling ratio paramemter, a certain ratio of relevant experiences are guaranteed to be replayed for DRL training.

are pre-defined for a mixture of the two types of experiences. The ratio $c = 1$ replays the relevant experiences only and $c = 0$ replays the irrelevant ones only.

The relevant experience set $R = \{e_t | e_t \in M, f_t = 1\}$ is retrieved from the memory using the tagging f_t, and the irrelevant set $I = \{e_t | e_t \in M, f_t = 0\}$. Concatenating the two sets, a batch B draws a total of b samples from the memory to train the DRL agent:

$$B = [e_t^{(r)}, e_t^{(i)}]_{e_t^{(r)} \sim R, e_t^{(i)} \sim I}$$

where $|e_t^{(r)}| = c * b$ and $|e_t^{(i)}| = (1 - c) * b$.

With this strategy, we enforce a certain proportion of relevant experiences to be replayed in every training step. The process of monitoring and training DRL agent with RER is summarized in Algorithm 1.

D. Normalize Perception

The perception is output of an activation function, Parametric Rectifier Linear Units (PReLU) [6] in this case. Therefore, the distribution of the perception on each dimension is not consistent for kernel similarity measurement. Therefore, we incrementally apply a standard normalization on each dimension of the perception in order to stabilize the similarity computation.

The perception is normalized with information from all previously seen perceptions but without storing actual perceptions. The normalization method accumulates the total number (N_p), the sum ($\boldsymbol{p_{sum}}$), and the sum of squared ($\boldsymbol{p^2_{sum}}$) of all the perceptions observed at time t and a set of current N_t perceptions p_t as:

$$\begin{cases} N_p = N_p + N_t \\ \mathbf{p}_{sum} = \mathbf{p}_{sum} + \sum_{p \in \mathbf{p}_t} p \\ \mathbf{p}^2_{sum} = \mathbf{p}^2_{sum} + \sum_{p \in \mathbf{p}_t} p^2. \end{cases}$$

The mean and standard deviation are computed as:

$$\boldsymbol{\mu} = \frac{\mathbf{p_{sum}}}{N_p} \quad \text{and} \quad \boldsymbol{\sigma} = \sqrt{\frac{\mathbf{p^2_{sum}} - 2 * \mathbf{p_{sum}} * \boldsymbol{\mu}}{N_p} + \boldsymbol{\mu}^2}.$$

With the accumulated normalization, the perception $\mathbf{p}$ is standardized to $\tilde{\mathbf{p}}$ as:

$$\tilde{\mathbf{p}} = \frac{\mathbf{p} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}. \tag{2}$$

IV. Experiments

A. Experiment Setup

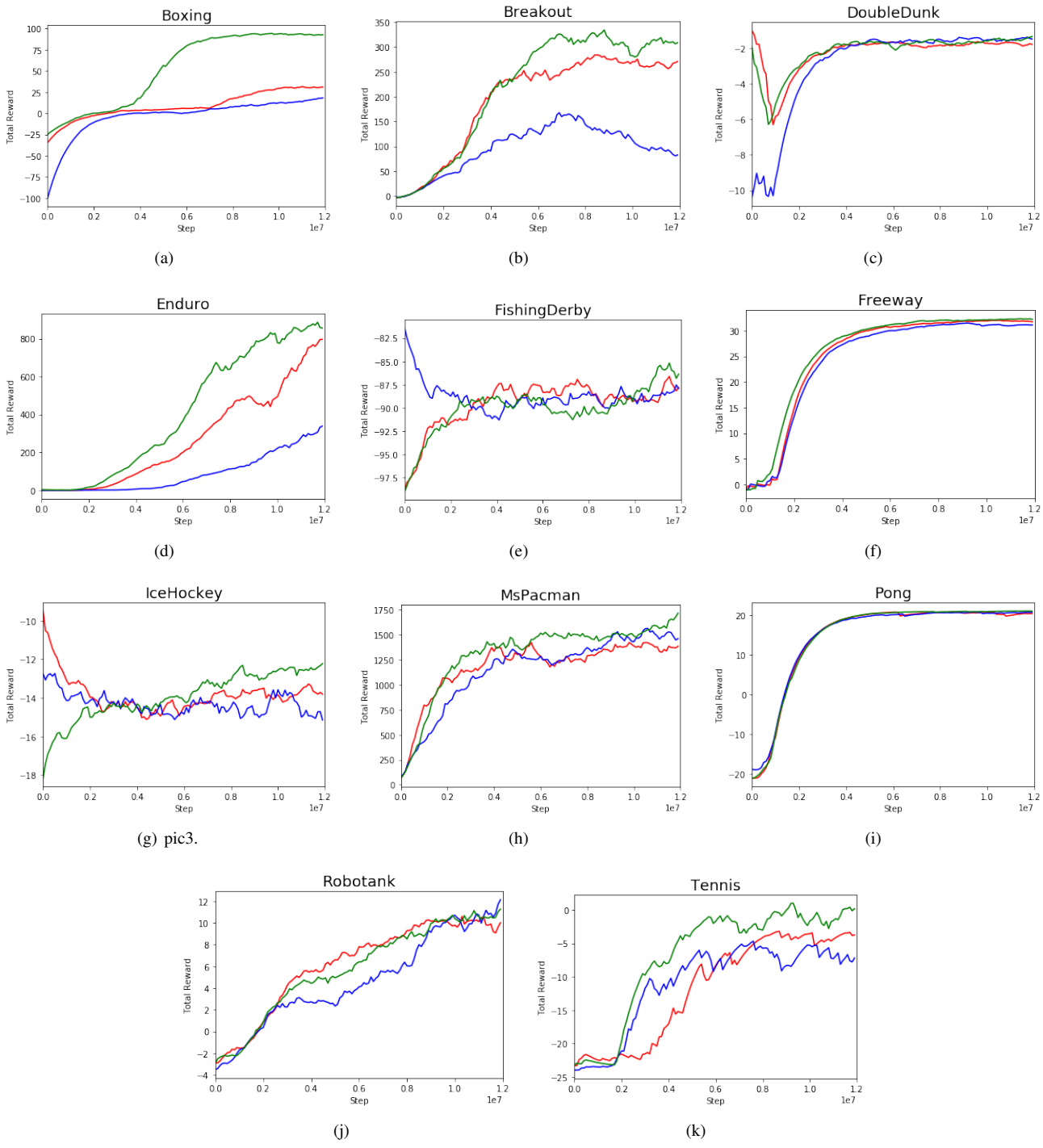
We examine the quality of the relevant experience replay with one of the state-of-the-art DRL algorithms, Double DQN [19]. All tests were run with the same training configuration for all the Atari game environments. We compare the performance of RER with Double DQN with two baselines, vanilla Double DQN that uses traditional experience replay and the one that uses prioritized experience replay.

a) Neural Networks: We slightly modified the architecture of Double DQN. We replaced the activation function to PReLU for both baselines and RER experiments. For computational efficiency, we use 256 hidden units in the last hidden layer.

b) Hyper-parameters: The discount factor γ is set to 0.99, and the learning rate is set to 10^{-4}. The number of steps between target network updates is $10,000$. We are optimizing the loss for DRL training with Adam optimizer [8]. All gradients are clipped between a range of $[-10, 10]$ to prevent gradient explosion problem when one gradient is too high or low.

The agent evaluates policy every 100K simulation steps and reports the average of the total reward from 30 episodes with random no-op actions from $1-30$ at the beginning of the game. By given random starting states, the agent has more robustness and generalization as the agent cannot rely on repeating a single memorized trajectory. The same random no-op action strategy is applied to the training phase as well. The mini-batch of $b = 64$ training samples are collected every $K_{train} = 4$ steps. The agent runs $T = 12$ million simulation steps.

The monitor is trained every $K_{monitor} = 256$ steps to ensure the monitor is up-to-date with the current policy. The



(a) (b) (c) (d) (e) (f) (g) pic3. (h) (i) (j) (k)

Fig. 3: Learning curves (average total reward of 30 episodes) for Double DQN with traditional experience replay in red, prioritized experience replay in blue, and relevant experience replay in green. Each curve corresponds to a single training run 12 million simulation steps.

gamma of perception and action for RBF kernel are set to $\gamma_p = 0.35$ and $\gamma_a = 1.0$ correspondingly.

To ensure rich exploration, we use ϵ-greedy with linearly decreasing ϵ from 1 to 0.1 for 1M simulation steps, from 0.1 to 0.01 for the next 5M simulation steps, and constantly 0.01 after that.

c) Environment: We tested randomly selected 11 different Atari game environments: Boxing, Breakout, Double Dunk, Enduro, Freeway, Ice Hockey, MsPacman, Pong, and Tennis. The environments are in NoFrameskip-v4 version from OpenAI Gym [2], which is similar environment done in [14]. We use a similar environmental setup to [14] except that the

terminal state gets the reward of -1 instead of 0 if the reward was initially 0, which indicates the agent lost of life.

d) Replay Buffer: Due to high memory usage and computation time for experience replay, we use a buffer with the size $N_m = 10^4$ for traditional experience replay, PER and RER to perform comparison. This will simulate the cases of how experience replay affects learning performance in complex problems that require large replay memory. The replay ratio c is set at 0.5 for balancing relevant and irrelevant experiences.

B. Empirical Results

Fig. 3 shows the raw average total rewards over 12 million steps of training time. In the presented nine Atari game experiments, relevant replay buffer shows improved performance, comparing to the traditional replay buffer and PER. Replaying with 50% ($c = 0.5$) of relevant experiences achieves faster learning speed (shorter time to reach the optimum) and higher asymptote (better solution with higher scores).

The RER does not immediately show an improvement in the early stage of the training because the agent needs to build up correct perception mapping through the convolutional layers. When the perception has been well established, the DRL-Monitor is capable of tagging the relevant experiences correctly to improve the agent's learning.

The efficiency of RER is established after the perception is well-formed. The process of formation typically takes 400K steps. The result in Fig. 3 shows a better improvement after 400K steps of RER compared to traditional experience replay and PER. The speed of learning curve is also improved with RER to reach a higher score.

Boxing, Breakout, Enduro, Ice Hockey, and Tennis show the improvements in term of learning speed after a well-established perception in Fig. 3. In Freeway and Pong, the problems are too simple to achieve further improvement in learning speed, but the RER does not slow down learning and achieves the equivalent learning speed and performance. In other environments, RER performs at least equivalent to or slightly better than traditional experience replay and PER due to the hardness of the problem with 10^4 buffer size.

In Fig. 4, we directly compare the trained agent with relevant experience replay with the two other baselines to see how much the proposed method improves quantitatively. The normalized improvement score of RER to traditional experience replay is computed comparing them with base human score as in [14]:

$$score_r = \frac{score_{relevant} - score_{human}}{score_{traditional} - score_{human}}. \quad (3)$$

This score tells how much better performance can be achieved by replacing traditional experience replay with relevant experience replay.

We also measure the traditional experience replay with PER using a similar metric formula. However, if the PER score is less than a human score, we reverse the formula and apply

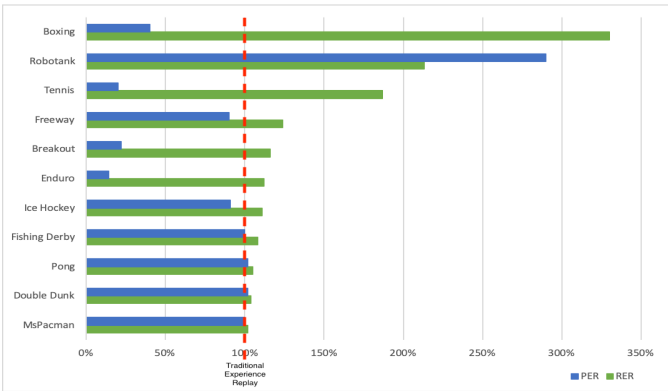


Fig. 4: Normalized improvement score comparison between RER, PER, and traditional experience replay in 11 different Atari game environments.

absolute value in the case of different sign when the human score is in between traditional experience replay and PER:

$$score_p = \left| \frac{score_{traditional} - score_{human}}{|score_{prioritized} - score_{human}|} \right|. \quad (4)$$

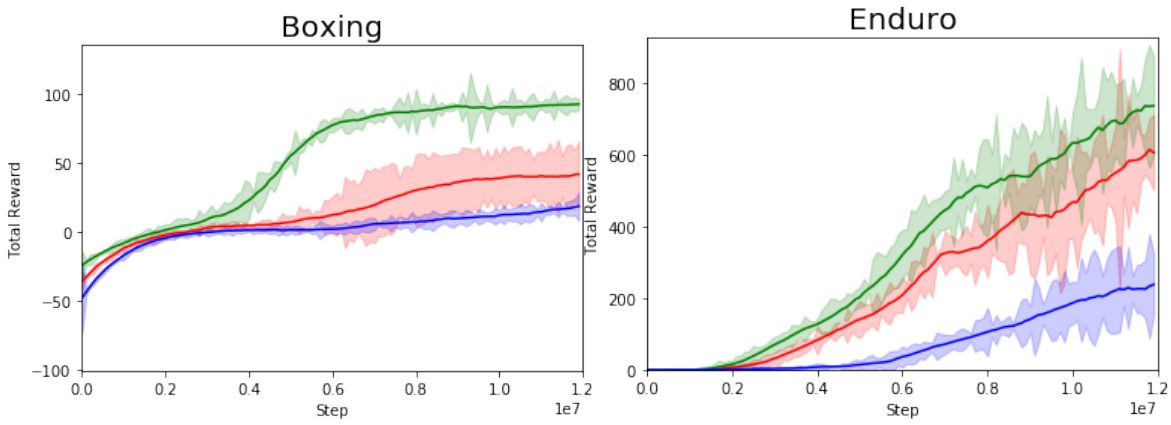
We observed a significant percentage improvement by applying relevant experience replay with an average increased of 47% across 11 Atari games that we evaluated. Notably, RER achieves higher improvements especially when the complexity of an environment grows. Fig. 4 depicts the comparison of the normalized scores in each Atari game with three comparison benchmarks after 12M simulation steps.

We observed a significant improvement in games where the immediate reward has both directions (negative and positive). They are shown with Boxing, Robotank, and Tennis environments. Pong and Freeway achieved maximum long term reward (thus the optimal policy), so they did not discover any better solution (or policy).

One directional immediate reward (only positive) also showed a noticeable improvement in Breakout, Enduro, and MsPacman. As we observe in Pong, exposure to too many irrelevant experiences results in instability of learning with traditional experience replay. The instability of Pong is shown by the red curve not always gets the maximum score 21.

Double Dunk and Fishing Derby are hard games for our agent with the environment setup of 10^4 buffer size. Therefore, RER was able to learn a slightly better solution than the one with traditional experience replay.

We also observed poor performance with the prioritized experience replay when we limit the size of the replay buffer. The performance decrements happen because of the gradient clipping. The probability of an experience to be drawn in PER depends on the temporal difference error. Therefore, when the gradient is clipped, the temporal difference error of a high error experience changes slowly. That makes the probability of the high error experience reduces slower. The buffer size of 10^4 is also not an ideal buffer size for PER. Since PER depends too much on the temporal difference error, small buffer size gets important experiences being pushed out faster. These two



(a) (b)

Fig. 5: Average and variance of 5 experiments on Boxing and Enduro environments. The color scheme is same as in Fig. 3 (traditional experience replay in red, prioritized experience replay in blue, and relevant experience replay in green).

reasons make PER sometimes performs worse than traditional experience replay. Especially the latter reason reveals the limitation of PER when the replay memory size is limited or when the environmental changes or complexity requires larger replay memory.

Fig. **??** show that mean and variance of learning performance in Boxing and Enduro environments for 5 runs. Because of limited computation resources, we were only able to fully run in these two environment. Small repetitions, however, show the similar trend to these results and previous results.

As we discussed before, we were able to observe the significantly improved asymptotic performance $p << 0.05$ from one-way ANOVA (p are 8.4×10^{-5} and 0.0012 in Boxing and Enduro respectly) in the two environments. Also, when we prioritize the experience (in green and blue curves), we can observe improved stability of learning learning while prioritized experience can be misguided when enough memory is not provided.

V. Conclusion

Experience replay has been one of the major components in reinforcement learning. There has been a number of techniques proposed to improve the data efficiency and stability learning for the experience replay. The process of determining relevant experiences is very important for storing problem in experience replay to mimic the psychological information processing from a human that remembers only relevant experiences when tackling a problem.

In this work, we introduced a simple, but novel method called Relevant Experience Replay that leverages DRL-Monitor to identify relevant experiences. The proposed approach examines the efficacy of DRL-Monitor as a tool for the decision-making problem of which experiences to store for experience replay. The empirical results verify the proposed approach. That is, DRL-Monitor is capable of identifying significant experiences to construct memory-efficient and effective experience replay model.

The proposed relevant experience replay is compatible with other DRL algorithms. Also, RER can be combined with HER and PER and is expected to further improve the learning performances. It will be natural next step for us to investigate the *storing* problem to realize a small experience replay buffer only with significant experiences based on DRL-Monitor's identification.

VI. Acknowledgement

This work was supported, in part, by funds provided by the University of North Carolina at Charlotte. The Titan Xp used for this research was donated by the NVIDIA Corporation.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay", In *Advances in Neural Information Processing Systems 30*, pages 5048–5058. Curran Associates, Inc., 2017.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym", *arXiv preprint arXiv:1606.01540*.

[3] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning", In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350, 2017.

[4] G. Dao, I. Mishra, and M. Lee, "Deep reinforcement learning monitor for snapshot recording", In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 591–598. IEEE, 2018.

[5] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading", *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[7] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay", *arXiv preprint arXiv:1803.00933*, 2018.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[9] M. Lee, "*Sparse Bayesian Reinforcement Learning*", PhD thesis, Colorado State University, 2017.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Hess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2016. ISSN 1935-8237. doi: 10.1561/2200000006.

[11] L.-J. Lin, "Self-improving reactive agents aased on reinforcement learning, planning and teaching", *Machine Learning*, 8(3):293–321, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699.

[12] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning", In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 372–382. IEEE, 2017.

[13] I. Mishra, G. Dao, and M. Lee, "Visual sparse Bayesian reinforcement learning: A framework for interpreting what an agent has learned", In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1427–1434. IEEE, 2018.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", *Nature*, 518(7540):529–533, 2015. ISSN 14764687. doi: 10.1038/nature14236.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay", *arXiv preprint arXiv:1511.05952*, 2015.

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

[17] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine", *Journal of Machine Learning Research*, 1:211–244, 2001. ISSN 15324435. doi: 10.1162/15324430152748236.

[18] M. E. Tipping, A. C. Faul, et al., "Fast marginal likelihood maximisation for sparse Bayesian models", In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

[19] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning", In *AAAI*, volume 16, pages 2094–2100, 2016.

[20] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay", *arXiv preprint arXiv:1611.01224*, 2016.

[21] J. Zhao, G. Qiu, Z. Guan, W. Zhao, and X. He, "Deep reinforcement learning for sponsored search real-time bidding", *arXiv preprint arXiv:1803.00259*, 2018.