A Deeper Look at Experience Replay

Shangtong Zhang, Richard S. Sutton
Dept. of Computing Science
University of Alberta
{shangtong.zhang, rsutton}@ualberta.ca

Abstract

Recently experience replay is widely used in various deep reinforcement learning (RL) algorithms, however in this paper we showcase that it is not as good as people think. To be more specific, experience replay will significantly hurt the learning process if the size of replay buffer is not well tuned. Although experience replay is a necessary component in modern deep RL algorithms to stabilize the network, we should be aware that the idea of experience replay itself is not as good as people think. The size of the replay buffer is an important hyper-parameter, which can significantly influence the performance and has unfortunately been underestimated in the community for a long time. In this paper we did a systematic empirical study of experience replay under various function representations. We showcase that a large replay buffer can significantly hurt the performance. Moreover, we propose a simple $\mathcal{O}(1)$ method to remedy the negative influence of a large replay buffer. We showcase its utility in both simple grid world and challenging domains like Atari games. Moreover, we visualize how a large replay buffer hurts the learning process.

1 Introduction

Experience replay enjoys a great success recently in the deep RL community and has become a new norm in many deep RL algorithms (Lillicrap et al. 2015; Andrychowicz et al. 2017). Until now it is the only method that can generate uncorrelated data for the online training of deep RL systems except the use of multiple workers (Mnih et al. 2016), which unfortunately changes the problem setting somehow. However in this paper, we showcase that experience replay is not as good as people always think. Some critical flaw of experience replay is hidden by the complexity of the deep RL systems, which explains the confusing phenomena that experience replay itself was proposed in the early age of RL, but it did not draw much attention when tabular methods and linear function approximation dominated the field. Experience replay not only provides uncorrelated data to train a neural network, but also significantly improves the data efficiency (Lin 1992;Wang et al. 2016), which is a desired property for many RL algorithms as they are often pretty hungry for data. Although algorithms in pre-deep-RL era do not need to care about how to stabilize a neural network, they do care data efficiency. If experience replay is a perfect idea, it should already be widely used in early ages. However unfortunately, to our best knowledge no previous work has shown what is wrong with experience replay.

Moreover, with the success of the Deep-Q-Network (DQN, Mnih et al. 2015), the community seems to have a default value for the size of the replay buffer, i.e. 10^6. For instance, Mnih et al. (2015) set the size of their replay buffer for DQN to 10^6 for various Atari games (Bellemare et al. 2013), after which Lillicrap et al. (2015) also set their replay buffer for Deep Deterministic Policy Gradient (DDPG) to 10^6 to address various Mujoco tasks (Todorov et al. 2012). Moreover, Andrychowicz et al. (2017) set their replay buffer to 10^6 in their Hindsight Experience Replay (HER) for a physical robot arm and Tassa et al. (2018) use a replay buffer with capacity as 10^6 to solve the tasks in the DeepMind Control Suite. In the aforementioned works, the tasks vary from simulation environments to real world robots and the function approximators vary from shallow fully-connected networks to deep convolutional networks. However they all use a replay buffer with same capacity. It seems to be robust under complex deep RL systems and nobody bothers to tune the replay buffer size. However when

we separate the experience replay from complex learning systems, we can easily find the agent is pretty sensitive to the size of the replay buffer. Some facts of replay buffer size are hidden by the complexity of the learning system.

Our first contribution is that we present a systematic evaluation of experience replay under various function representations, i.e. tabular case, linear-function approximation and non-linear function approximation. We showcase that both a small replay buffer and a large replay buffer can heavily hurt the learning process. In other words, the size of the replay buffer, which has been under-estimated by the community for a long time, is an important task-dependent hyper-parameter that needs careful tuning. Some facts of experience replay are hidden by the complex modern deep RL systems and the idea of experience replay itself is not as good as people think.

Another contribution is that we propose a simple method to remedy the negative influence of a large replay buffer, which requires only $\mathcal{O}(1)$ extra computation. To be more specific, whenever we sample a batch of transitions, we add the latest transition to the batch and use the corrected batch to train the agent. We refer to this method as combined experience replay (CER) in the rest of this paper.

It is important to note that experience replay itself is not a complete learning algorithm, it has to be combined with other algorithms to form a complete learning system. In our evaluation, we consider the combination of experience replay with Q-learning (Watkins 1989), following the DQN paradigm.

We perform our evaluation and showcase the utility of CER in both small toy task, e.g. Grid World and Mountain Car, and large scale challenging domains, e.g. the Lunar Lander and Atari games. Moreover, we visualize how a large replay buffer hurts the learning process.

2 Related Work

CER can be treated inaccurately as a special case of prioritized experience replay (PER, Schaul et al. 2015). In PER, Schaul et al. (2015) proposed to give the latest transition a largest priority. However PER is still a stochastic replay method, which means giving the latest transition a largest priority does not guarantee it will be replayed immediately. Moreover, it is important to note that PER and CER are aimed to solve different problems, i.e. CER is designed to remedy the negative influence of a large replay buffer while PER is designed to replay the transitions in the buffer more efficiently. To be more specific, if the replay buffer size is set properly, we do not expect CER can further improve the performance however PER is always expected to improve the performance. Although there is a similar part to CER in PER, i.e. giving a largest priority to the latest transition, PER never shows how that part interacts with the size of the replay buffer and whether that part itself can make a significant contribution to the whole learning system. Furthermore, PER is an $\mathcal{O}(\log N)$ algorithm with fancy data structures, e.g. a sum-tree, which significantly prevents it from being widely used. However CER is an $\mathcal{O}(1)$ plug-in, which needs only little extra computation and engineer effort.

Liu and Zou (2017) did a theoretical study on the influence of the size of the replay buffer. However their analytical study focused on an ordinary differential equation model, and their experiments focused only on toy domains. We present a systematic study on a variety of value function representations and various challenging domains.

Experience replay can be interpreted as a planning method, because it is comparable to Dyna (Sutton 1991) with a look-up table. However, the key difference is that Dyna only samples *states* and *actions*, while experience replay samples *full transitions*, which may be biased and potentially harmful.

There are also successful trials to eliminate experience replay in deep RL. The most famous one is the Asynchronous Advantage Actor-Critic method (Mnih et al. 2016), where experience replay was replaced by parallelized workers. The workers are distributed among processes, and different workers have different random seeds. As a result, the data collected is still uncorrelated.

3 Algorithms

Experience replay was first introduced by Lin (1992). The key idea of experience replay is to train the agent with the transitions sampled from the a buffer of previously experienced transitions. A transition is defined to be a quadruple (s, a, r, s'), where s is the state, a is the action, r is the received reward after executing the action a in the state s and s' is the next state. At each time step, the current transition is added to the replay buffer and some transitions are sampled from the replay buffer to train the agent. There are various sampling strategies to sample transitions from the replay buffer, among which uniform sampling is the most popular one. Although there is also prioritized sampling (Schaul et al. 2015), where each transition is associated with a priority, it always suffers from $\mathcal{O}(\log N)$ time complexity. So we therefore constrict our evaluation in uniform sampling.

We compared three algorithms: Q-Learning with online transitions (referred to as Online-Q, Algorithm 1), Q-Learning with experience replay (transitions for training only from the buffer, referred to as Buffer-Q, Al-

gorithm 2) and Q-Learning with CER (referred to as Combined-Q, Algorithm 3). Online-Q is the primitive Q-Learning, where the transition at every time step is used to update the value function immediately. Buffer-Q refers to DQN-like Q-Learning, where the current transition is not used to update the value function immediately. Instead, it is stored into the replay buffer and only the sampled transitions from the replay buffer are used for learning. Combined-Q uses both the current transition and the transitions from the replay buffer to update the value function at every time step.

4 Testbeds

We use four tasks to evaluate the aforementioned algorithms. A grid world, the mountain car, the lunar lander and the Atari game Pong. Figure 1 elaborates the tasks.

Our first task is a grid world, the agent is placed at the same location at the beginning of each episode (*S* in Figure 1(a)), and the location of the *goal* is fixed (*G* in Figure 1(a)). There are four possible actions $\{$*Left*, *Right*, *Up*, *Down*$\}$, and the reward is -1 at every time step, implying the agent should learn to reach the *goal* as soon as possible. Some fixed walls are placed in the grid worlds, and if the agent bumps into the wall, it will remain in the same position.

Our second task is the mountain car task from OpenAI gym (Brockman et al. 2016), we adjust the timeout of an episode to 500 from 200 as our preliminary experiments show that a Q-learning based agent can hardly solve the task with timeout set to 200.

The third task is the lunar lander task in Box2D (Catto 2011), which is a typical task that needs careful exploration. Negative rewards are continuingly given during the landing so the algorithm can easily get trapped in a local minima, where it avoids negative rewards by doing nothing after certain steps until timeout.

The last task is the Atari game Pong. It is important to note our evaluation is aimed to study the idea of experience replay. We are not going to study how the experience replay interacts with a convolutional network. To this end, it is better to use an accurate state representation in the game rather than try to learn the representation during an end-to-end training. We therefore use the ram of the game as the state rather than the raw pixels, which is a vector of length 256. Each element of the vector is an integer valued in $[0, 255]$.

5 Evaluation

Different mini-batch size has different computation complexity, as a result, throughout our evaluation we do not vary the batch size within a single task and use a mini-batch of fixed size 10 unless otherwise specified, i.e. we sample 10 transitions from the replay buffer at each time step. For CER, we only sample 9 transitions and the mini-batch consists of the sampled 9 transitions and the latest transition. The behavior policy is a ϵ-greedy policy with $\epsilon = 0.1$. We plot the on-line training progression for each experiment unless otherwise specified, i.e. we plot the episode return against the number of training episodes during the on-line training.

5.1 Tabular Case

Among the four tasks, only the grid world is compatible with tabular methods.

In the tabular representation, the value function q is represented by a look-up table. The initial values for all state-action pairs are set to 0, which is an optimistic initialization (Sutton 1996) to encourage exploration. The discount factor is set to 1.0, and the learning rate is 0.1.

Figure 2 (a - c) shows the training progression with timeout set to 200. In Figure 2(a), Online-Q solves the task in about $5,000$ episodes. In Figure 2(b), it is interesting to note that smallest replay buffer works best. Buffer-Q with buffer capacity 40 solves the task in only $2,000$ episodes, which is faster than Online-Q. However when we increase the buffer capacity to 60, the green line suggests that the agent fails to find the best solution, although it does learn something sub-optimal. When we keep increasing the buffer capacity, as suggested by the read line and the purple line (which is unfortunately hidden by the red line), the agent fails to learn anything. We observed same phenomena in Figure 2(c) for Combined-Q, i.e. the larger the buffer is, the worse the performance is. This is because a larger replay buffer contains more outdated data, which will bias the agent more. Moreover when we compare Figure 2(b) and Figure 2(c), we can clearly see the importance of the on-line transition. For a buffer with capacity 40, adding the on-line transition increases the learning speed. For a buffer with capacity 60, adding the on-line transition not only increases the learning speed but also improves the asymptotic performance. Buffer-Q can only find a sub-optimal policy while Combined-Q succeeds in finding the optimal policy. For a buffer with capacity 100, Buffer-Q does not learn anything however Combined-Q ends up with a sub-optimal policy. However for a buffer with capacity 10^6, adding the on-line transition does not make a huge difference. This suggests the benefit of adding on-line tran-

Algorithm 1: Online-Q

Initialize the value function q
while *not converged* **do**
  Get the initial state s
  while s *is not the terminal state* **do**
    Select an action a according to a ϵ-greedy policy derived from q
    Execute the action a, get the reward r and the next state s'
    Update the value function q with (s, a, r, s') following the Q-learning update rule
    $s = s'$
  end
end

Algorithm 2: Buffer-Q

Initialize the value function q
Initialize the replay buffer $\mathcal{M}$
while *not converged* **do**
  Get the initial state s
  while s *is not the terminal state* **do**
    Select an action a according to a ϵ-greedy policy derived from q
    Execute the action a, get the reward r and the next state s'
    Store the transition (s, a, r, s') into the replay buffer $\mathcal{M}$
    Sample a batch of transitions $\mathcal{E}$ from $\mathcal{M}$
    Update the value function q with $\mathcal{E}$ following the Q-learning update rule
    $s = s'$
  end
end

Algorithm 3: Combined-Q

Initialize the value function q
Initialize the replay buffer $\mathcal{M}$
while *not converged* **do**
  Get the initial state s
  while s *is not the terminal state* **do**
    Select an action a according to a ϵ-greedy policy derived from q
    Execute the action a, get the reward r and the next state s'
    Store the transition $e = (s, a, r, s')$ into the replay buffer $\mathcal{M}$
    Sample a batch of transitions $\mathcal{E}$ from $\mathcal{M}$
    Update the value function q with $\mathcal{E}$ and e following the Q-learning update rule
    $s = s'$
  end
end

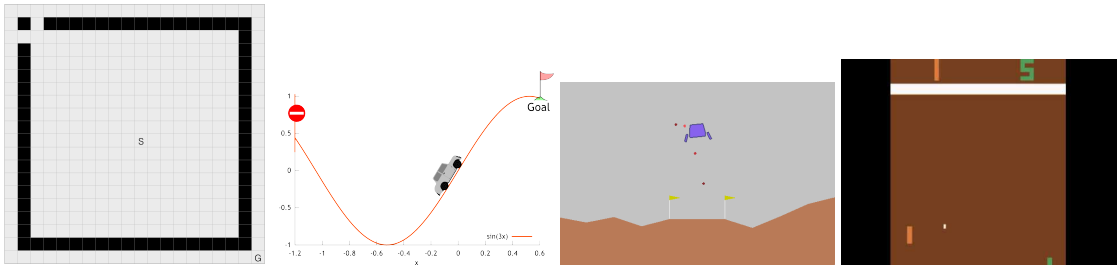


Figure 1: From left to right: the grid world, Mountain Car, Lunar Lander, Pong

sition is limited. The larger the buffer is, the weaker the help of the on-line transition is.

As a tabular function representation does not suffer from the stabilization issue, the only benefit of a replay buffer is the improved data efficiency. However we have to note that the buffer data is outdated, so the improved data efficiency comes with a price. Table 1 shows the real steps and total processed transitions before the agent solves the task. In terms of real steps, Buffer-Q and Combined-Q learns faster than Online-Q. However, when it comes to total processed transitions, Online-Q learns faster, which denotes that the transition sampled from the replay buffer is worse than the real transition. So when getting new transitions is not so expensive and stabilization is not an issue, learning with real transitions still saves overall computation resource compared with sampled outdated transitions.

In the grid world task, the timeout of an episode determines the difficulty of the task. With a large timeout, an agent can always reach the goal via randomness of the exploratory behavior policy. So something useful can be learned at each episode. When we increase the timeout from 200 to $5,000$, Figure 2(d - f) shows all the three algorithms find the optimal policy. However we can still observe that a larger replay buffer hurts the learning speed in Buffer-Q and adding the on-line transition increases the learning speed for all the buffer sizes.

5.2 Linear Function Approximation

We still only consider the grid world task in the linear function approximation case.

We use linear function approximation with tile coding (Sutton and Barto 1998), which is done by the tile coding software [1] with the number of tilings set to 8. We set the initial weight parameters to 0 to encourage exploration. The discount factor is set to 1.0 and the learning rate is $0.1/8 = 0.125$. The results are summarized in Figure 3. Figure 3(b) demonstrates that a larger replay buffer hurts the learning speed in Buffer-Q. Compared with Figure 3(c), it is clear that adding the on-line transition significantly improves the learning speed, especially for a large replay buffer. The results are similar to what we observed in tabular function representation.

5.3 Non-linear Function Approximation

We use a single hidden layer network as our non-linear function approximator. We apply *Relu* (Nair and Hinton 2010) nonlinearity over the hidden units and the output units are linear to compute the state-action value. With neural networks as the function approximator, Buffer-Q is almost the same as DQN. Thus we also employs a target network to gain stable update targets following Mnih et al. (2015). Note that our preliminary experiments show that random exploration at the beginning stage and a decayed exploration rate do not help the learning process in our tasks.

For the grid world task, we use 50 hidden units and for the other tasks we use 100 hidden units. In the grid world task, we use a one-hot vector to encode the current position of the agent and in Pong we normalize the state into $[0, 1]$. For Pong we set the mini-batch size to 32 instead of 10. We always employ a RMSProp optimizer (Tieleman and Hinton 2012) while the initial learning rates vary among tasks. We use 0.001 for Mountain Car and Lunar Lander and use 0.01 and 0.00025 for the grid world and Pong respectively. For the grid world task, we set the timeout of an episode to 500 steps.

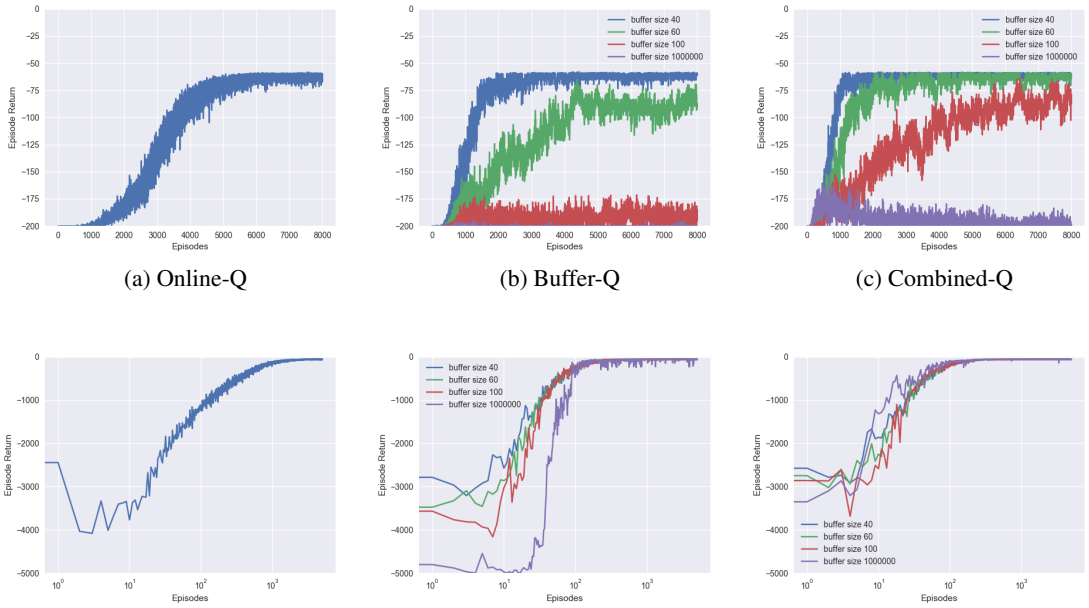
Our preliminary experiments show that Online-Q does not work with non-linear function approximation in our tasks, so we do not present the results of Online-Q in this section.

Figure 4 shows the learning progression in various domains. Different from tabular and linear function representation, where a larger replay buffer always hurts the learning speed, we observe in Figure 4(c) and 4(e) that a buffer with capacity 10^4 performs best, which is a medium buffer size. We hypothesize that there is a trade-off between the data quality and data correlation. With a smaller replay buffer, data tends to be more fresh however they are highly temporal correlated, while training a neural network always needs i.i.d. data. With a larger replay buffer, the sampled data tends to be uncorrelated, however they are more outdated, so the agent gets more biased. Comparing the performance of Buffer-Q and Combined-Q in the four tasks, it is obvious that adding the on-line transition significantly improve the performance of the agent with a large replay buffer, e.g. the grid world agents with buffer capacity 10^5 and 10^6, the Mountain Car agent with buffer capacity 10^6, the Lunar Lander agent with buffer capacity 10^5 and the Pong agent with buffer capacity 10^6.

6 Visualization

We visualize some of the experiments to show how a large replay buffer hurts the learning process. As it is easy to compute the true state value function v^* of the grid world task under the optimal policy, we base our visualization on the grid world task with a non-linear function approximator. We use $Q(s, a)$ to demote the estima-

[1] `http://incompleteideas.net/sutton/tiles/tiles3.html`



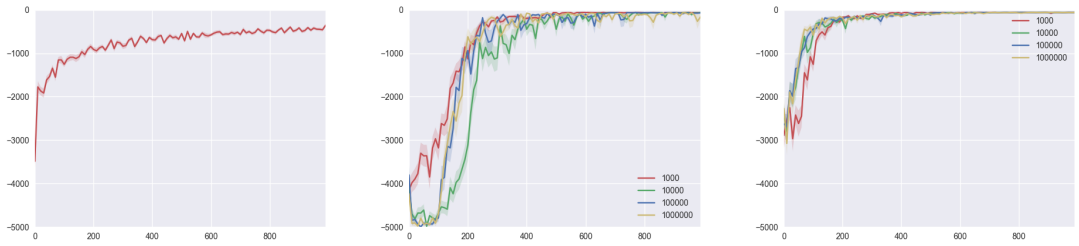
(a) Online-Q (b) Buffer-Q (c) Combined-Q

(d) Online-Q (e) Buffer-Q (f) Combined-Q

Figure 2: Training progression in tabular case on the grid world. In (a)(b)(c) timeout is set to 200 while in (d)(e)(f) timeout is set to 5000 and the x-axis is transformed to *log*-scale to better distinguish the difference. In (b), the purple curve is hidden by the red one. The results are averaged over 30 independent runs.

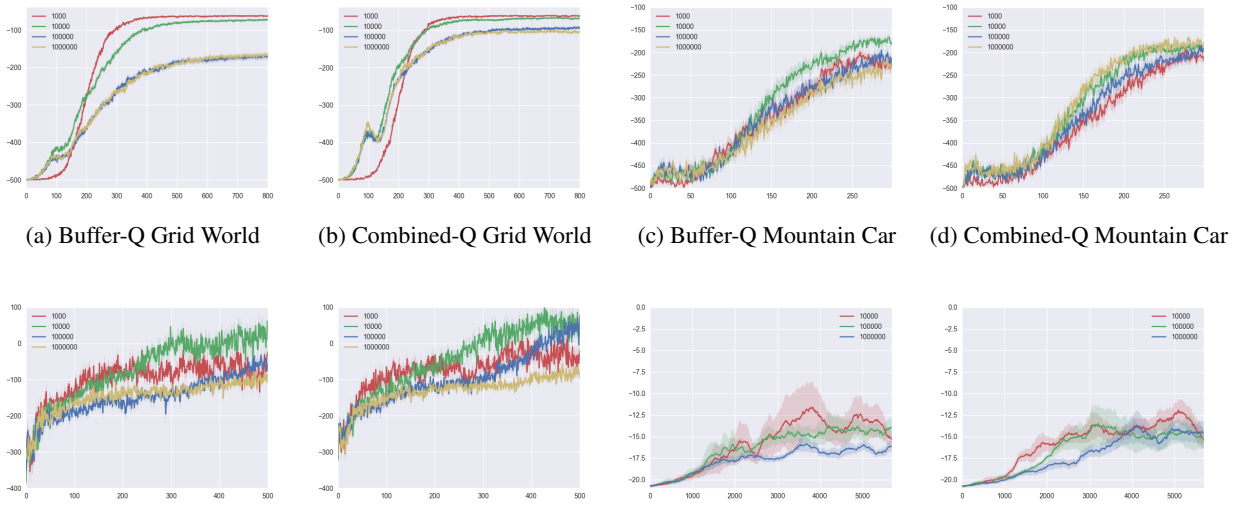
	replay buffer size	real steps	processed transitions	successful trials
Online-Q	N/A	715,822	715,822	30
Buffer-Q	40	223,909	2,239,094	30
Buffer-Q	60	393,245	3,932,459	30
Buffer-Q	100	76,293	762,933	2
Buffer-Q	10^6	N/A	N/A	0
Combined-Q	40	157,151	1,728,665	30
Combined-Q	60	210,363	2,314,000	30
Combined-Q	100	393,048	4,323,531	30
Combined-Q	10^6	N/A	N/A	0

Table 1: Steps before solving the grid world task with timeout set to 200. The agent solves the this task if the average return of recent 50 episodes is larger than -70. Each algorithms had 30 trials.



(a) Online-Q (b) Buffer-Q (c) Combined-Q

Figure 3: Training progression of linear function approximator on the grid world task with timeout set to 5000. All the plots are averaged over 30 independent runs. For better display, the curves are smoothed by a sliding window of size 30.



(a) Buffer-Q Grid World (b) Combined-Q Grid World (c) Buffer-Q Mountain Car (d) Combined-Q Mountain Car

(e) Buffer-Q Lunar Lander (f) Combined-Q Lunar Lander (g) Buffer-Q Pong (h) Combined-Q Pong

Figure 4: Tasks with non-linear function approximator. Standard error is plotted. (a - b) Training progression averaged over 500 runs. (c - d) Training progression averaged over 60 runs. (e - f) Training progression averaged over 30 runs. (g - h) Test progression averaged over 10 runs. We perform 10 deterministic test episodes every 10 training episodes and use the mean return of the test episodes as the final performance. It is expected that the agent does not solve Pong, as it may be to difficult to approximate the state-value function with a single-hidden-layer network.

tion of the state-action value, thus the estimation of the state value is

$$V(s) = \max_a Q(s, a)$$

We first visualize the badness of the sampled mini-batches. Recall that an agent learns from a transition $t = (s, a, r, s')$ by pushing its estimation $Q(s, a)$ to $r + \gamma V(s')$. So intuitively, we can say a transition is good for the agent if its estimation $V(s')$ is close to the true value $v^*(s')$. Following this intuition, we define the badness of a transition as

$$b(t) \doteq |v^*(s') - V(s')|$$

The smaller the badness is, the more the agent can learn. So for a mini-batch $\{t_0, \ldots, t_N\}$, we draw the histogram of $\{b(t_0), \ldots, b(t_N)\}$ to visualize its badness. Figure 5 shows the badness of the sampled batches in a single run for Buffer-Q agents with buffer size 10^4 and 10^5. It is clear that with a larger replay buffer, it is harder to get a *good* mini-batch.

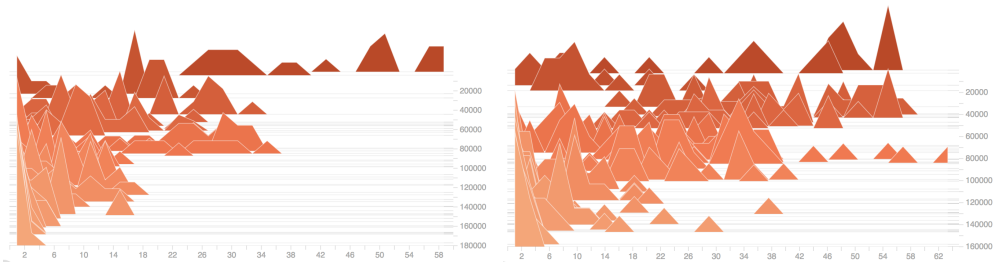
We also visualize the quality of the transitions in the replay buffer. We expect the transitions in the replay buffer to be uniformly distributed as training a neural network needs i.i.d. data. We use $v^*(s_i)$ to identify a transition $t_i = (s_i, a_i, r_i, s'_i)$. Assume at time step k, the replay buffer consists of $\{t_0, \ldots, t_N\}$, we then plot the histogram of $\{v^*(s_i), \ldots, v^*(s_N)\}$ to investigate the quality of the transition distribution in the replay buffer at that time step. Figure 6 shows the distribution of the replay buffer data at different training steps along a single run of Buffer-Q with the replay buffer size 10^4 and 10^5. With the small replay buffer, the transitions get uniformly distributed soon under our measurement. However with a large replay buffer, the transition distribution is still skewed at the end of training. Note that in Figure 6(b), the total training steps is 150% of the replay buffer size, so old transitions have been allowed to be replaced.

7 Conclusion

Experience replay can improve data efficiency and stabilize the training of a neural network, however it does not come for free. It can constantly provides outdated data, which may heavily bias the learned value function, resulting in worse data distribution in the replay buffer. This flaw is hidden by the complexity of model deep RL systems. In fact experience replay is not as good as people think. This negative effect is partially controlled by the size of replay buffer, which is shown in this paper to be an important task-dependent hyper-parameter but has been under-estimated by the community for a long time. PER is a promising approach addressing this issue, however it often comes with $\mathcal{O}(\log N)$ complexity and non-negligible extra engineer effort. We propose CER, which is similar to a component in PER but only requires $\mathcal{O}(1)$ extra computation, and showcase it can significantly remedy the negative influence of a large replay buffer.

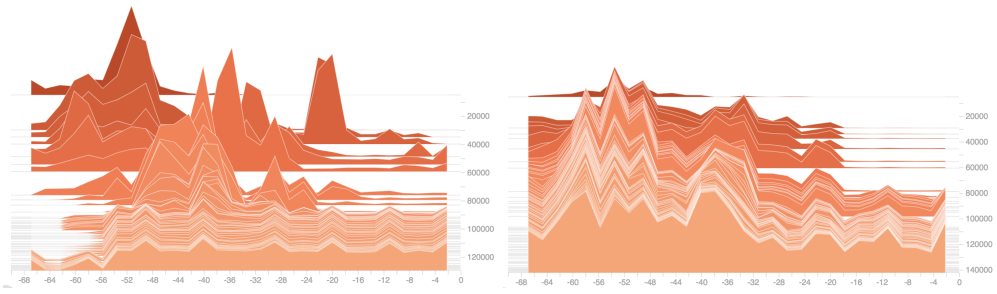
Acknowledgements

The authors thank Kristopher De Asis and Yi Wan for their thoughtful comments.



(a) Buffer-Q (1e4) (b) Buffer-Q (1e5)

Figure 5: Visualization of the badness of the sampled batches along training. The y-axis is the total training steps and each slice along the x-axis represents a histogram showing the badness of the sampled batch at the corresponding training step.



(a) Buffer-Q (1e4) (b) Buffer-Q (1e5)

Figure 6: Visualization of the distribution of the replay buffer data. The y-axis is the total training steps and each slice along the x-axis represents a histogram showing the replay buffer data distribution.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *arXiv preprint arXiv:1707.01495*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Catto, E. (2011). Box2d: A 2d physics engine for games.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-H. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3/4):69–97.

Liu, R. and Zou, J. (2017). The effects of memory replay in reinforcement learning.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE.

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

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.