A Deeper Look at Experience Replay

Shangtong Zhang
Dept. of Computing Science
University of Alberta
shangtong.zhang@ualberta.ca

Richard S. Sutton
Dept. of Computing Science
University of Alberta
rsutton@ualberta.ca

Abstract

Experience replay plays an important role in the success of deep reinforcement learning (RL) by helping stabilize the neural networks. It has become a new norm in deep RL algorithms. In this paper, however, we showcase that varying the size of the experience replay buffer can hurt the performance even in very simple tasks. The size of the replay buffer is actually a hyper-parameter which needs careful tuning. Moreover, our study of experience replay leads to the formulation of the Combined DQN algorithm, which can significantly outperform primitive DQN in some tasks.

1 Introduction

Experience replay was first introduced by Lin (1992) and applied in the Deep-Q-Network (DQN) algorithm later on by Mnih et al. (2015). The 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. Most deep RL algorithms sample the transitions uniformly from the replay buffer, while Schaul et al. (2015) introduced the prioritized experience replay, where preference was given to certain transitions. In this paper, we limit our study on experience replay with uniform sample. Experience replay is often coupled with some other algorithms, so it is impossible to investigate experience replay without a concrete algorithm. As DQN is the first paradigm which combines deep learning and reinforcement learning, we focus on the combination of Q-Learning (Watkins 1989) and experience replay in our study. Moreover, all of the deep RL algorithms with experience replay only train the agent with the transitions sampled from the replay buffer. The newly generated transition is not immediately trained on. In our study, we also explore the possibility combining the sampled transitions with the latest transition. We investigate the combination of experience replay and Q-Learning systematically in tabular, linear function approximation and non-linear function approximation settings.

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*, while experience replay samples *full transitions*, which may be biased and potentially harmful. The reason for experience replay's success in deep RL still remains unclear. A common hypothesis is that experience replay provides temporally uncorrelated (i.i.d) data to train the neural network on and it can counteract catastrophic forgetting by regularly updating on old transitions. However, this hypothesis needs to be more carefully studied. In this paper, we showcase that even in very simple tasks, experience replay can hurt the performance. The size of the replay buffer is actually a hyper-parameter which should be carefully tuned.

2 Related Work

Experience replay appears in many deep RL algorithms as a standard component, e.g. Trust Region Policy Optimization (Schulman et al. 2015), Proximal Policy Optimization (Schulman et al. 2017) and Deep Deterministic Policy Gradient (Lillicrap et al. 2015). It also has many variants. Andrychowicz et al. (2017) incorporated *goal* into the experience replay to learn as much as possible from the sampled transitions, resulting in the Hindsight Experience Replay. Foerster et al. (2017) brought *fingerprint* into experience replay to deal with multi-agent learning. Jaderberg et al. (2016) proposed skewed sampling to address the issues of sparse rewards. Experience replay was also used in the actor-critic method to improve sample efficiency (Wang et al. 2016).

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.

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 on general-purposed tasks, e.g. MountainCar and CartPole. We present a systematic study on a variety of value function representations, and we designed simpler tasks to develop more intuition.

3 Algorithms

We compared three algorithms: Q-Learning with online transitions (referred to as Online-Q, **Algorithm 1**), Q-Learning with buffer transitions (referred to as Buffer-Q, **Algorithm 2**) and Q-Learning with both online and buffer transitions (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. With neural networks as the function approximator, Buffer-Q is almost the same as DQN. In that case, we also apply necessary tricks following Mnih et al. (2015), e.g. random exploration at the beginning stage, decayed exploration rate and target network, to make it as stable as possible.

```
Initialize the value function $q$
Get the initial state $s$
while $s$ is not the terminal state do
    Select an action $a$ according to the current value function $q$
    Execute the action $a$, get the reward $r$ and the next state $s'$
    Update the value function $q$ with ($s$, $a$, $r$, $s'$)
    $s = s'$
end
```

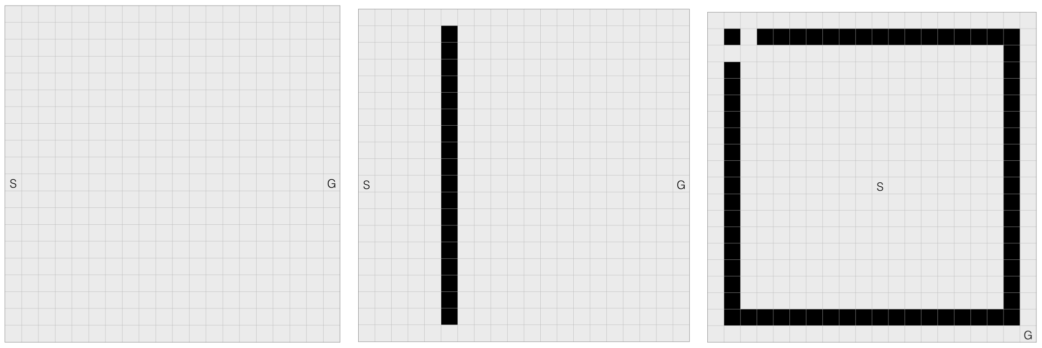
Algorithm 1: An episode of Online-Q.

```
Initialize the value function $q$
Initialize the replay buffer $\mathcal{M}$
Get the initial state $s$
while $s$ is not the terminal state do
    Select an action $a$ according to the current value function $q$
    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 $\mathcal{E}$
    $s = s'$
end
```

Algorithm 2: An episode of Buffer-Q.

```
Initialize the value function $q$
Initialize the replay buffer $\mathcal{M}$
Get the initial state $s$
while $s$ is not the terminal state do
    Select an action $a$ according to the current value function $q$
    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 $e$
    $s = s'$
end
```

Algorithm 3: An episode of Combined-Q.



(a) Easy Grid World (b) Medium Grid World (c) Difficult Grid World

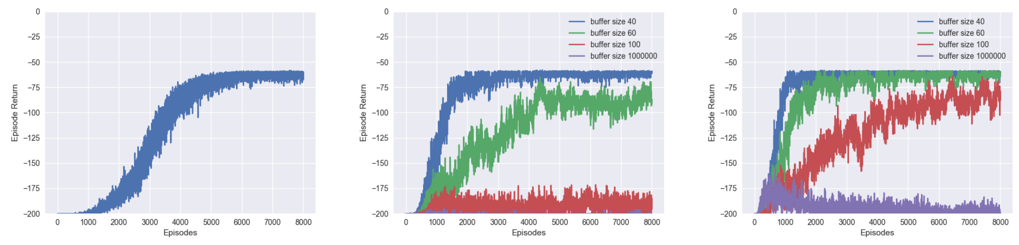
Figure 1: *S* represents the start position and *G* is the goal. Black squares are walls.

4 Experiments

Our testbeds include three grid worlds. The agent is placed at the same location at the beginning of each episode, and the location of the *goal* is fixed. 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. We set the discount factor to 1.0 for each experiment, and we sampled a batch of 10 transitions from the replay buffer at every time step. As our preliminary experiments showed that larger batch size only has minor improvement on the learning speed, we did not tune the batch size in the reported experiments. The grid worlds have a predefined timeout, i.e. the episode ends automatically after certain time steps.

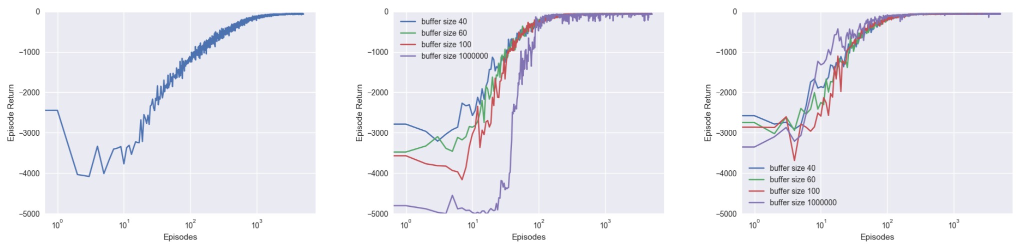
The first task is the *Easy Grid World* without any walls, the second task is the *Medium Grid World* and the last one is the *Difficult Grid World*. All the grid worlds are 20×20 and elaborated in **Figure 1**. At the early stage, the agent just walks around the start position, so the replay buffer is filled with many trivial transitions. However, the key to solving the task is to learn from some critical transitions, e.g. a transition to the goal or a transition to a well learned position. We designed the three tasks to investigate whether the experience replay method can get rid of the influence of the trivial transitions at the early stage and how the size of the replay buffer influences the performance.

We plotted the return at each episode, and all the results of the grid worlds were averaged over 30 independent runs. To make the differences clearer at early episodes, the episode-axis is in *log* scale for some of the plots. We selected reasonable maximum episodes for each algorithm so that it can converge on a stable solution. As we got similar trends in the results across the different grid world difficulties, we only present the plots for the *Difficult Grid World*. Plots for the *Easy Grid World* and the *Medium Grid World* can be found in **Appendix**.



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

Figure 2: Training progression of tabular case on *Difficult Grid World* with timeout set to 200. In (b), the purple curve is hidden by the red one.



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

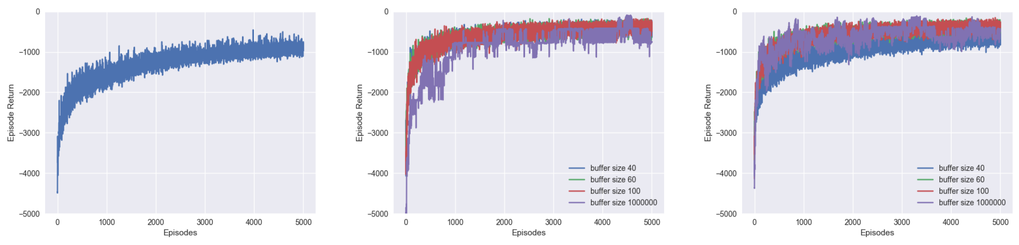
Figure 3: Training progression of tabular case on *Difficult Grid World* with timeout set to 5000.

4.1 Tabular Case

In the tabular case, the value function q was represented by a look-up table. The initial values for all state-action pairs were set to 0, which is an optimistic initialization (Sutton 1996) to encourage exploration. The behavior policy was an ϵ-greedy policy with $\epsilon = 0.1$, and the learning rate was 0.1. **Figure 2** shows the training progression with timeout set to 200. **Figure 2(b)** and **Figure 2(c)** demonstrates that the small replay buffer, where trivial transitions are more easy to get replaced, can benefit learning. When we increased the capacity of the replay buffer, the learning slowed down. Comparing **Figure 2(b)** and **Figure 2(c)**, we can see the online update does help correct the negative effect of the trivial transitions sampled from the replay buffer. **Table 1** shows the real steps and total processed transitions before the agent solved the task. In our experiments, Buffer-Q and Combined-Q processed 10 and 11 transitions per real time step. In terms of real steps, Buffer-Q and Combined-Q learned faster than Online-Q. However, when it comes to total processed transitions, Online-Q learned faster, which denotes that the transition sampled from the replay buffer is not guaranteed to be always

	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 *Difficult Grid World* 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 4: Training progression of linear function approximator on *Difficult Grid World* with timeout set to 5000.

better than the real transition. Although the sampled transitions from large replay buffer are more likely to be uncorrelated, they, however, turned out to hurt the learning process in this task. **Figure 3** shows the training progression with timeout set to 5000. Although the differences among different replay buffer sizes are smaller, we can still draw same conclusion.

4.2 Linear Function Approximation

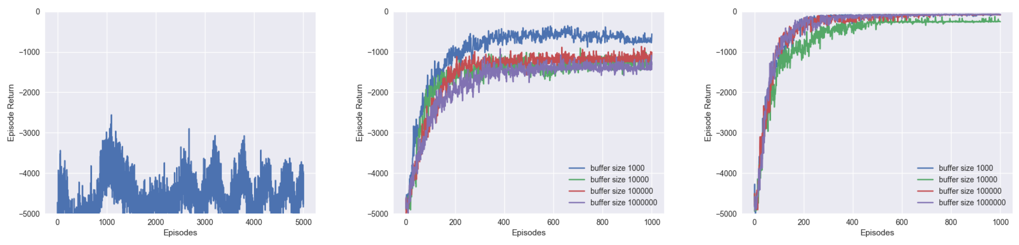
We used linear function approximation with tile coding (Sutton and Barto 1998), which was done by the tile coding software [1] with the number of tilings set to 8. We also set the initial weight parameters to 0 to encourage exploration. The learning rate was $0.1/8 = 0.125$ and we used an ϵ-greedy policy with $\epsilon = 0.1$. The results are summarized in **Figure 4**. **Figure 4(b)** demonstrates that the learning process benefits from a small replay buffer, and extremely large replay buffer has trouble solving the task within 5,000 episodes. Comparing with **Figure 4(c)**, it's clear that combing the online transition with transitions from replay buffer can significantly speed up learning, even with a large replay buffer.

4.3 Non-Linear Function Approximation

We used a single *ReLU* (Nair and Hinton 2010) hidden layer network with 50 hidden units as the non-linear function approximator, and the output layer is linear. We used binary encoding to encode the state (x, y) into a binary vector of length 10. (As the grid world is 20×20, 5 bits are enough for each coordinate.) The optimizer was *RMSProp* (Tieleman and Hinton 2012) with an initial learning rate of 0.01. 1,000 random exploration steps were carried out at the beginning with the initial exploration rate ϵ set to 1.0, and ϵ was linearly decreased to 0.1 over 10,000 time steps. All of these techniques aims to help with exploration, as we do not have optimistic initialization for neural networks. We updated the target network every 1,000 time steps. For Combined-Q, after sampling a batch of transitions like DQN, we added the current transition into the sampled batch and used the "corrected" batch to update the value network. The idea here is the contribution of the current transition should be equal to one single transition in the sampled batch rather than the whole batch. But apparently it depends on the size of the batch. We also refer DQN following this training protocol as Combined DQN (CDQN). **Figure 5(a)** shows that Online-Q can hardly learn the task with non-linear function approximator. One possible reason is the catastrophic forgetting that the network only focuses on fitting the latest transitions without memorizing the old ones. **Figure 5(b)** shows replay buffer does help get rid of catastrophic forgetting. In this case, the agent did learn something, but far from solving the task. And similarly to tabular and linear case, large replay buffer may actually hurt the learning. **Figure 5(c)** demonstrates that the combination of online update and update with buffer transitions indeed helps learning, in terms of both the learning speed and the asymptotic performance. In CDQN, the sampled transitions from replay buffer are in charge of avoiding catastrophic forgetting while the online transition helps correct the bias of the "bad" transitions sampled from the replay buffer, resulting in a more efficient learning together.

We also evaluated DQN and CDQN in the Atari game *Pong* (Bellemare et al. 2013). We used exactly same hyper-parameters and network architectures as reported by Mnih et al. (2015). In this case,

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



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

Figure 5: Training progression of non-linear function approximator on *Difficult Grid World* with timeout set to 5000.

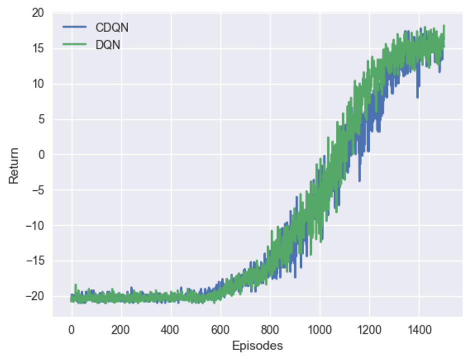


Figure 6: Training progression of DQN and CDQN in *Pong*. The curves are averaged over 5 independent runs.

DQN and CDQN achieved same performance level (**Figure 6**). So CDQN is actually a promising alternative of DQN. In general, CDQN has same computation complexity and can achieve at least the same performance level as DQN.

5 Discussion

The community should not take experience replay for granted. Although in popular benchmarks, e.g. Atari games and *MuJoCo* (Todorov et al. 2012), using a large replay buffer seems not to be harmful, for some tasks, however, the size of the replay buffer appears to be a critical hyper-parameter, which needs to be carefully tuned. It can be set according to certain property of the task. For example, in the three grid worlds, setting the buffer size to the length of a typical episode seems a good choice. With a large replay buffer, the critical transitions can be easily hidden by the trivial transitions, result in the inefficiency of the learning process. In conclusion, different tasks may need replay buffer of different size, and large replay buffer may also result in poor performance. Moreover, we propose CDQN, combing the online transition and transitions from the replay buffer. In general, CDQN has the same computation complexity as DQN and it will not hurt the performance in existing popular benchmarks. More important, CDQN can significantly outperform primitive DQN in some tasks. So we should consider CDQN a promising alternative of primitive DQN, and similar techniques can easily be applied in many other experience replay based deep RL algorithms.

6 Acknowledgment

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

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.

Foerster, J., Nardelli, N., Farquhar, G., Torr, P., Kohli, P., Whiteson, S., et al. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

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*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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.

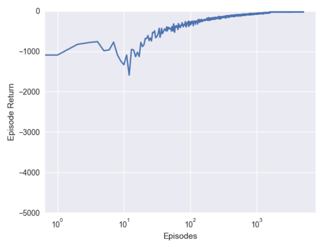
Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

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.

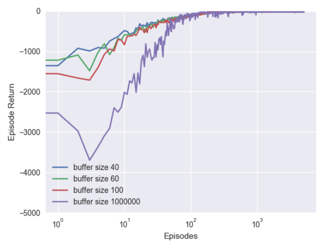
Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

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

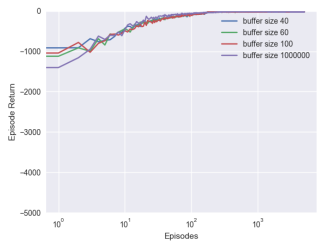
A Results on *Easy Grid World*



(a) Online-Q

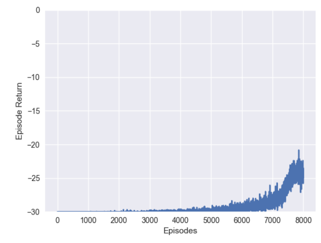


(b) Buffer-Q

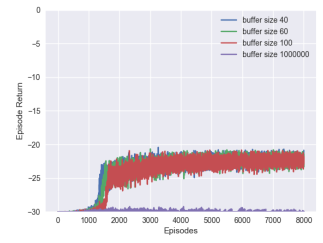


(c) Combined-Q

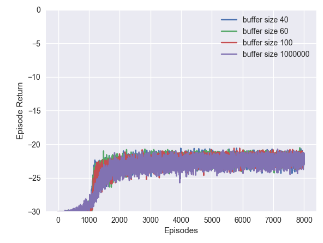
Figure 1: Training progression of tabular case on *Easy Grid World* with timeout set to 5000.



(a) Online-Q

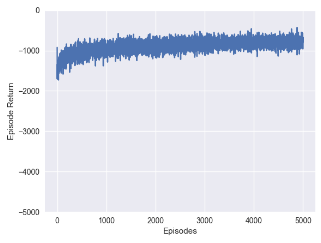


(b) Buffer-Q

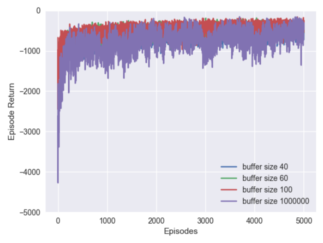


(c) Combined-Q

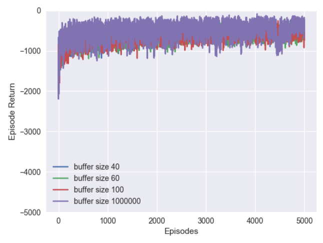
Figure 2: Training progression of tabular case on *Easy Grid World* with timeout set to 30.



(a) Online-Q

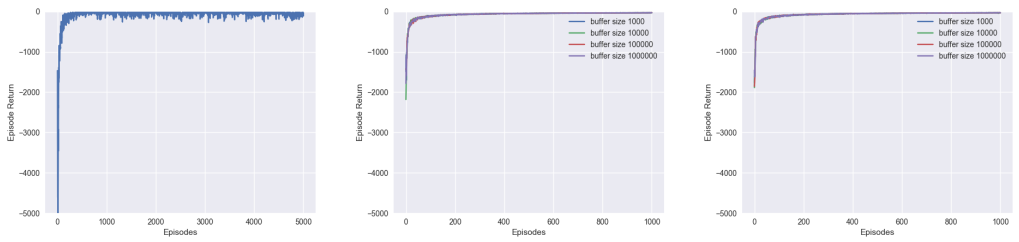


(b) Buffer-Q



(c) Combined-Q

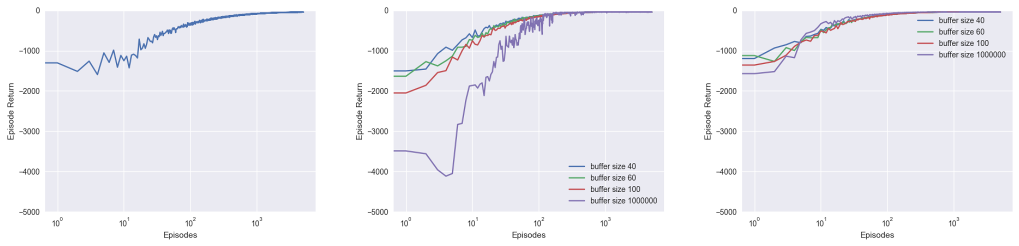
Figure 3: Training progression of linear function approximation on *Easy Grid World* with timeout set to 5000.



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

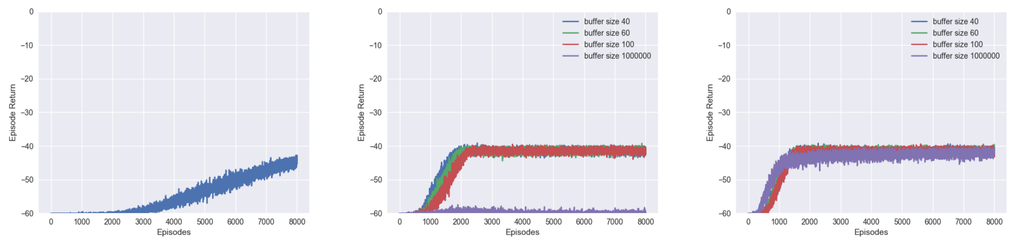
Figure 4: Training progression of non-linear function approximation on *Easy Grid World* with timeout set to 5000.

B Results on *Medium Grid World*



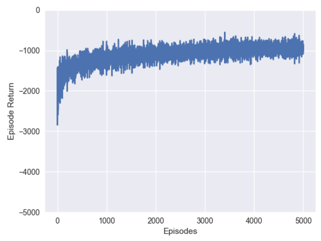
(a) Online-Q (b) Buffer-Q (c) Combined-Q

Figure 5: Training progression of tabular case on *Medium Grid World* with timeout set to 5000.

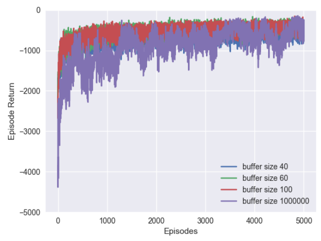


(a) Online-Q (b) Buffer-Q (c) Combined-Q

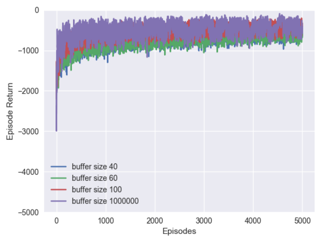
Figure 6: Training progression of tabular case on *Medium Grid World* with timeout set to 60.



(a) Online-Q

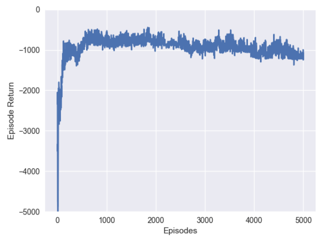


(b) Buffer-Q

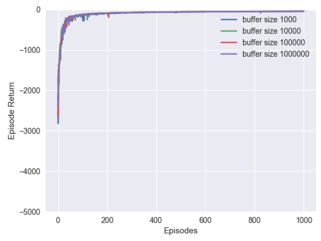


(c) Combined-Q

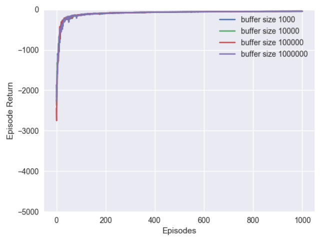
Figure 7: Training progression of linear function approximation on *Medium Grid World* with timeout set to 5000.



(a) Online-Q



(b) Buffer-Q



(c) Combined-Q

Figure 8: Training progression of non-linear function approximation on *Medium Grid World* with timeout set to 5000.