Experience Filtering for Robot Navigation using Deep Reinforcement Learning

Phong Nguyen[1], Takayuki Akiyama[1] and Hiroki Ohashi[2]
[1]*R&D Group, Hitachi Ltd., 1-280, Higashi-Koigakubo, Kokubunji-shi, Tokyo, Japan*
[2]*European R&D Group, Hitachi Europe GmbH, Kaiserslautern, Germany*

Keywords: Reinforcement Learning, Experience Replay, Similarity, Distance, Experience Filtering.

Abstract: We propose a stochastic method of storing a new experience into replay memory to increase the performance of the Deep Q-learning (DQL) algorithm, especially under the condition of a small memory. The conventional standard DQL method with the Prioritized Experience Replay method attempts to use experiences in the replay memory for improving learning efficiency; however, it does not guarantee the diversity of experience in the replay memory. Our method calculates the similarity of a new experience with other existing experiences in the memory based on a distance function and determines whether to store this new experience stochastically. This method leads to the improvement in experience diversity in the replay memory and better utilization of rare experiences during the training process. In an experiment to train a moving robot, our proposed method improved the performance of the standard DQL algorithm with a memory buffer of less than 10,000 stored experiences.

1 INTRODUCTION

A reinforcement learning algorithm is considered a prominent solution for robot navigation. A state-of-the-art reinforcement learning algorithm is the standard Deep Q-learning (DQL) algorithm recently proposed by Mnih (Mnih et al., 2015). It uses an artificial neural network to map state spaces to Q-values of actions, which will be used by an agent to select the best action in a given state. This algorithm stores all the experiences of the agent in a replay memory and the artificial neural network learns by sampling the stored experiences. For the agent to perform well after training, positive experiences (examples of success) and negative experiences (examples of failures) are theoretically essential to acquire and store in the replay memory. The balance of the replay memory determines the efficiency of the training process. When positive or negative experiences are difficult to acquire, the replay memory becomes imbalanced. An artificial neural network may not learn well under such conditions because the experiences are sampled randomly for learning. Shaul presented the Prioritized Experience Replay (PER) method to prioritize the sampling process to obtain experiences with high temporal difference error (TD-error) and improve the training process (Schaul et al., 2015). However, this method does not improve the balance of the replay memory. In a scenario in which some experiences are extremely difficult to acquire and the replay memory is relatively small, such experiences are quickly removed due to the first-in-first-out (FIFO) mechanism. Such rare experiences should be retained in the replay memory long enough for the artificial neural network to reduce the TD-error and learn well. Therefore, the diversity of the replay memory plays an important role in the efficiency of the training process.

In the Internet-of-Things (IoT) era, the DQL algorithm is expected to be applied to embedded devices, which usually have small memories. In our research, we aimed at applying the DQL algorithm under such a limited condition. Therefore, in this paper, we propose a stochastic method of storing a new experience into the replay memory to improve the performance of the DQL algorithm. Our method calculates the similarity of a new experience with other existing experiences in the memory based on a distance function and determines whether to store this new experience stochastically.

We achieved higher learning efficiency by observing the average reward of the agent after the training process is completed. In an experiment to teach a robot to move (by avoiding obstacles and approach hu-

Nguyen, P., Akiyama, T. and Ohashi, H.
Experience Filtering for Robot Navigation using Deep Reinforcement Learning.
DOI: 10.5220/0006671802430249
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 2*, pages 243-249
ISBN: 978-989-758-275-2

mans for interaction) using the DQL algorithm, our proposed method improved the performance of the DQL algorithm with a memory buffer of less than 10,000 stored experiences by almost 20% in the robot's average reward.

2 RELATED WORK

To make a robot move without it hitting obstacles and approach a certain goal, a path-planning algorithm and rule based algorithm have been investigated (Fahimi, 2008) (LaValle, 2006). While the path-planning algorithm is negatively affected by a dynamic and constantly changing environment because it always requires re-planning when the environment changes, a rule-based algorithm usually cannot cover all situations. Moreover, these algorithms are not adaptable to different types of environments. With rein-forcement learning, however, a robot can learn how to execute better actions through trial-and-error processes without any prior history or knowledge (Kaelbling et al., 1996) (Sutton and Barto, 1998). Thus, rule definitions and map information are not needed in advance to train a moving model.

Reinforcement learning has recently received a lot of attention in many research fields (Mirowski et al., 2016) (Sadeghi and Levine, 2016) (Jaderberg et al., 2016) (Wang et al., 2016). Mnih et al. (Mnih et al., 2015) (Mnih et al., 2013) reported a method of combining an artificial neural network with a Q-learning algorithm to estimate the action value. The method was tested on teaching a machine to play Atari games from the input of raw image data. In games like Montezumas Revenge, the machine failed to learn from the replay memory because the small number of positive experiences was subservient. In another paper, Shaul et. al attempted to utilize useful experiences by using a prioritized sampling method (Schaul et al., 2015). Using TD-error as the metric to prioritize the experience sampling process, this method can learn more efficiently from the rare experiences in the replay memory. However, this method cannot lead to a diversity of experiences in the memory, and rare and useful experiences can be pushed out of the memory quickly when the memory is small.

To ensure the replay memory is well diversified and rare and useful experiences in the memory can be kept longer, our proposed method uses a filtering mechanism in the experience storing process. The filtering mechanism works to filter out the new experiences that are similar to many existing experiences in the memory and only store the new experiences that are different from other existing experiences. Even with the limited size of replay memory, the neural network can learn more effectively from a diversified and balanced replay memory. Our proposed method is compatible with the conventional PER method, and we can combine them to achieve a superior performance in term of achieving higher reward after the training process is completed.

Other research on robotic systems that traverse indoor environments using deep reinforcement learning for collision avoidance focuses on transfering knowledge from simulation world to real world like in (Sadeghi and Levine, 2016) , or using recurrent neural network to encode the history of experience like in (Jaderberg et al., 2016). None of the above mentioned research has tackled the problem of limited replay memory size.

3 STANDARD DQL METHOD AND PER METHOD

Deep reinforcement learning represents the Q-function with a neural network, which takes a state as input and outputs the corresponding Q-values of actions in that state. Q-values can be any real values, which makes it a regression task that can be optimized with a simple squared error loss (Mnih et al., 2015):

$$L = \frac{1}{2}[\underbrace{r + \gamma \cdot \max_{a'} Q(s',a')}_{\text{target}} - \underbrace{Q(s,a)}_{\text{prediction}}]^2 \quad (1)$$

An experience of a DQL agent is a tuple of ¡state, action, reward, next state¿, hereby known as $< s,a,r,s' >$. The DQL agent at state s, takes the action a, and move on to the next state s', in which it receives a reward r. In each iteration, a transition experience $< s,a,r,s' >$ is stored in a replay memory. To train the artificial neural network, uniformed random samples from the replay memory are used instead of the most recent transition. This breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum. PER method modifies the standard DQL method from random sampling to prioritized sampling (Schaul et al., 2015). Experiences with high TD-error (the artificial neural network does not estimate the Q-values accurately), are most likely get sampled. There are two variations with PER method: proportional PER and rank-based PER. In proportional PER method, an experience has the probability of getting sampled proportional with its TD-error. On the other hand, in rank-based PER method, an experience has the probability of getting sampled negatively proportional with its TD-errors rank in the replay memory.

4 PROPOSED METHOD

In our method, instead of storing all the experiences in the replay memory, each new experience is evaluated on whether it should be stored or dropped out. We present three filtering mechanisms for our method to filter new experiences.

4.1 Greedy Experience Filtering (EF)

When a new experience is observed, we compute the distances of the new experience with other experiences in the memory. The distance between two experiences $E1 < s1, a1, r1, s1' >$ and $E2 < s2, a2, r2, s2' >$ is defined as follows:

- Distance between s_1 and s_2 (Bernstein et al., 2001):

$$d(s_1, s_2) = \text{Euclidean Distance } (s_1, s_2) \text{ normalized in range } [0, 1] \quad (2)$$

- Distance between a_1 and a_2:

$$d(a_1, a_2) = \begin{cases} 0 & \text{if } a_1 = a_2 \\ 1 & \text{if } a_1 \neq a_2 \end{cases} \quad (3)$$

- Distance between r_1 and r_2:

$$d(r_1, r_2) = \begin{cases} 0 & \text{if } r_1 = r_2 \\ 1 & \text{if } r_1 \neq r_2 \end{cases} \quad (4)$$

- Distance between s_1' and s_2' (Bernstein et al., 2001):

$$d(s_1', s_2') = \text{Euclidean Distance } (s_1', s_2') \text{ normalized in range } [0, 1] \quad (5)$$

- Distance d of $E1 < s1, a1, r1, s1' >$ and $E2 < s2, a2, r2, s2' >$:

$$d(E_1, E_2) = \min\big[1, \frac{1}{2} d(s_1, s_2) + \frac{1}{2} d(s_1', s_2') + d(a_1, a_2) + d(r_1, r_2)\big] \quad (6)$$

After computing all the distances of the new experience with other experiences in the replay memory, we make a stochastic decision to store the new experience in the replay memory with the following probability:

$$P(E_{new}) = D(E_{new}, E_{memory}) = \frac{1}{N} \sum_{i=1}^{N} d(E_{new}, E_i) \quad (7)$$

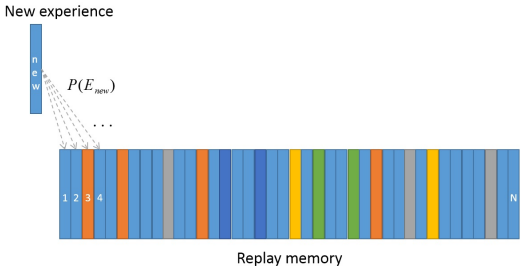

Figure 1: Probability of storing new experience.

where N is the number of experiences in the replay memory at the point before storing a new experience. In formula (6), the distance between two experiences are capped maximum at 1 because later in formula (7) we will use the average distances to assign the probabilities of storing the new experience. Capping the the distance at 1 leads a more stable performance. The way we design the distance d between 2 experiences also ensures that when the agent is at a same state, but takes different action or gains different rewards, the algorithm would consider these 2 experiences very different from each other.

Distance D shows how different a new experience is to the existing memory. If D is high, the replay memory is more likely to store the new experience. However, if D is low, it is not likely to store it. Fig. 1 illustrates the process of computing the probability to store a new experience. In this figure, similar experiences (experiences with small D) are represented in a similar color (e.g. blue), while rare experiences (experiences with large D) are represented in different colors (e.g. red, green, orange). When the replay memory buffer is filled and it is decided that the new experience is stored, we need to drop an old experience in the replay memory to make space for the new experience. To do so, we compute the sum of the distances between an experience and other experiences in the replay memory and choose the experience with the lowest distance sum:

$$Experience\,ID\,to\,drop = \arg\max_i \left(\sum_{j=1}^{N} d(E_i, E_j) \right) \quad (8)$$

By dropping the experience with the lowest distance sum, we maximize D among experiences. Therefore, the experiences in the memory are well diversified, and the rare and useful experiences will be kept inside the memory.

4.2 FIFO EF

The difference of FIFO EF and Greedy EF is when we have to drop an old experience to add a new experience. Instead of choosing the lowest distance sum of an experience with other experiences, we simply drop the oldest experience in the replay memory. The reason to do this is to avoid over-fitting in training the neural network to a certain set of experiences without having more updated experience. When only dropping the experience with the lowest distance sum in the memory, some experiences in the memory are never dropped. The neural network can only learn well with these old experiences; hence, it cannot generalize to other situations. By implementing FIFO EF, we still achieve our purpose of trying to achieve a better balance of experiences in the replay memory by a filtering process, while providing the neural network with new data along the training process and having the neural network generalize better.

4.3 Combination of FIFO EF and PER

Our proposed method can work compatibly with the conventional PER method. In each training iteration, we use FIFO EF method to decide whether to store a new experience into the memory or not, and then proceed with sampling method presented in Shaul et. al.'s paper. Our method ensures the diversity of replay memory, which would be impossible to achieve if storing whatever experiences the agent observes. Prioritized experience sampling method in a diversed replay memory makes more sense than in a monotonous replay memory.

In addition, our method can retain the rare experience inside the replay memory, in comparsion to the conventional way of storing experience because each new experience is stored based on a probability, which makes the rate of storing expriences is slower than storing all experiences. For that reason, prioritizing experience with high TD-error can resample rare experiences more frequently and improve the learning efficiency.

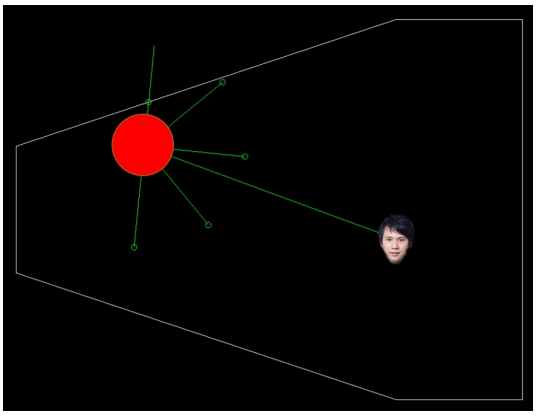

Figure 2: Simulated environment.

5 EXPERIMENTAL SETUP

The simulation environment we used contains a layout of walls (called a map). Moving objects including a robot and human (in the role of a customer) who can only move within the boundary of the map. The robot has distance sensors that sense the distance to objects in front of it. There are five distance sensors, which sense five angles equally divided in front of the robot. The maximum distance a distance sensor can sense is 4 m. With a human detection sensor (e.g. like a camera), the robot can detect the targeted customer, the distance between it and the customer, and the angle of the customer in relation to it.

We designed a reward system so that the robot can receive a positive reward +1 when it is successfully approaches and faces the customer and the distance between it and the customer is less than 1 m. However, it receives a negative reward of -1 when it hits the walls. It will then be reset to a random position on the map.

The deep neural networks we used for action selection and training are fully connected networks, which contain an input layer, two hidden layers, and an output layer. The input layer contains:

- five distance values from the distance sensor of the robot,
- current information about the robot: current speed, and current angular speed,
- the estimated distance to the customer,
- the angle α to the customer, which is represented by two values: $\sin\alpha$ and $\cos\alpha$, and
- the relative speed of the robot toward the customer.

And all these input values are normalized to a suitable range: the distance values, the current speed of robot, the distance to the customer have a range from 0 to 1, and the current angular speed of the robot, the relative speed of the robot toward the customer have the range from -1 to 1. Each hidden layer contains 256 nodes. The output layer contains 9 nodes corresponding to Q-values of 9 actions for the robot. Each action is a combination of the robot making two decisions about whether to change its speed and angular velocity (decrease the speed, maintain speed, increase the speed, increase the angular velocity to the left,

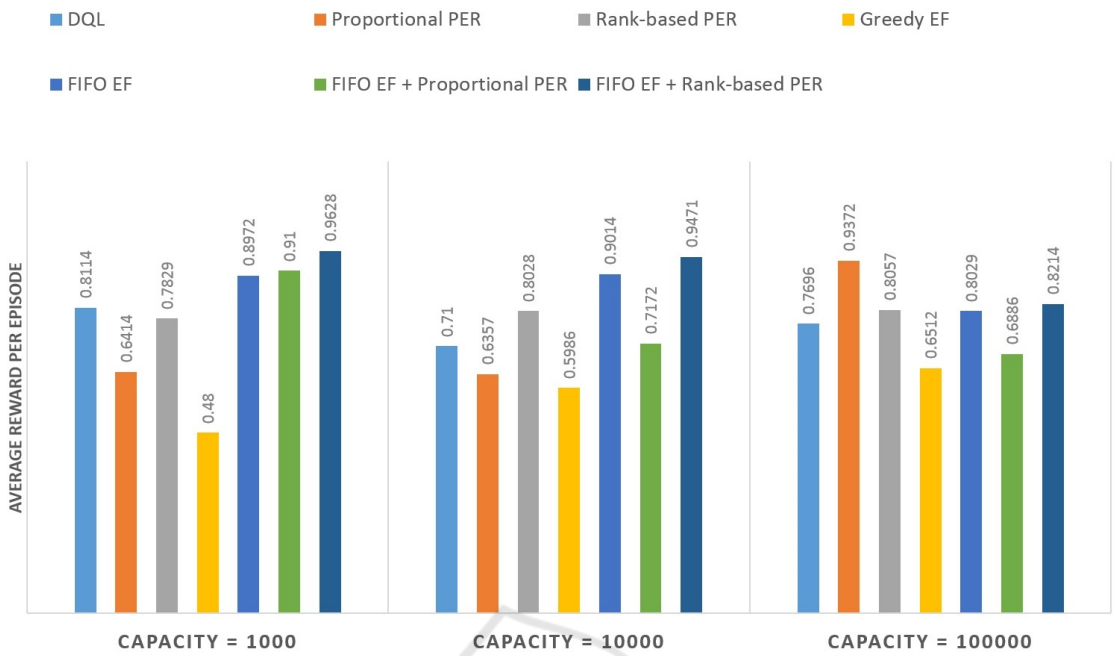

Figure 3: Average reward per episode after training.

keep the same angular velocity, and increase the angular velocity to the right). In other words, nine actions of the robot are a product of three actions in changing speed and three actions in changing angular velocity.

We trained the robot for 10^6 iterations with a replay memory capacity of 10^3, 10^4, and 10^5. We used a tunnel-like map with a customer moving inside (the customer moved randomly). Fig. 2 illustrates the simulated training environment. The red circle represents the robot. Five short green lines and small circles on them represent the distance sensors and their sensing values. The picture of a face represents a human customer. The long green line represents the distance between the robot and the customer. In the evaluation process (after the training process is completed), we let the robot start at a random position on the map, at a fixed given time, and observed whether it could avoid obstacles and approach the customer successfully. An episode of evaluation starts when the robot starts and ends when either the robot meets the customers, hits the walls, runs out of time (we defined as in 200 iterations). We compared the performance of the robot by using a conventional standard DQL method (Mnih et al., 2015), PER method (both proportional and rank-based) alone (Schaul et al., 2015), our proposed method with FIFO EF, that with greedy EF, that with FIFO EF and proportional PER, and that with FIFO EF and rank-based PER.

6 RESULTS

The performance of the robot was evaluated based on the rate it could approach the customer successfully without hitting obstacles. The rate the robot hit the obstacles is also a reference of its performance. We observed the robot in 1000 episodes and calculated the average reward for each episode. Fig. 3 shows the results of a testing phase with the different methods in different memory size settings. Our proposed method with FIFO EF and that with FIFO EF and rank-based and proportional PER outperformed the DQL method and the PER methods alone when the memory size was 10^3, 10^4. When the replay memory capacity reached 10^5, our proposed methods with FIFO EF and that with FIFO EF and rank-based and proportional PER still performed better than the DQL method and about the same with the PER methods alone. However, our proposed method with greedy EF did not achieve the expected performance and had the lowest performance.

When the capacity size was 10^3, the average reward per episode the robot could receive with our proposed method, except with greedy EF, were higher than those with the DQL method. Fig. 4 shows the improvement (or deterioration) of our proposed method compared with that of the DQL method. With our method with FIFO EF and rankbased PER , we could increase performance by 18.66% after training.

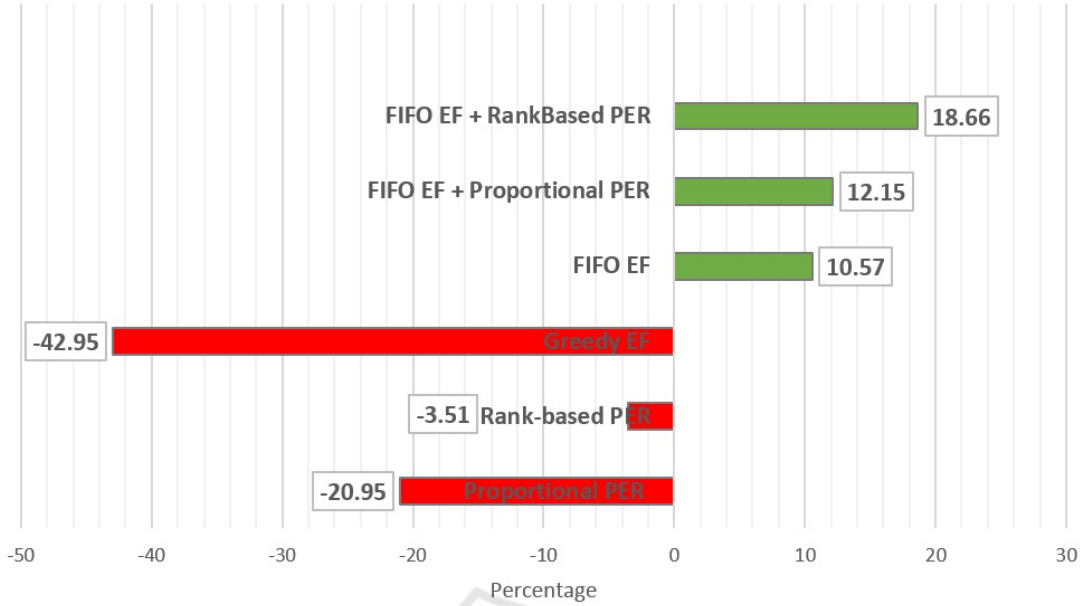

Figure 4: Robot's performance compared with standard DQL method (capacity = 10^3).

In other words, the robot could receive 18.66% more rewards per episode. This implies it successfully approached the customer more often and hit less obstacles after training by using our proposed method.

We break down the robot's performance of receiving rewards into successfully approaching the customer and hitting obstacles. The rate of finding and approaching a customer after training was 97.57% while that of only hitting obstacles was 1.1%, as shown in Fig. 5. The top left of the chart is the desired region for ideal performance after training: approached customer 100% and hit walls 0%.

Our results have demonstrated that with a smaller memory size (10^3 experience instances compared to 10^6 experience instances), our proposed method can give the DQL algorithm a good performance on two learning tasks: avoiding obstacles and approaching a goal (in this case, a customer), while conventional methods cannot perform well.

7 DISCUSSION

In this chapter, we discuss the strength and weakness of our method based on the results. The strength of our method is providing more diversity in experiences; hence, breaking the dominance of similar experiences in the memory, which causes neural networks to be unable to learn efficiently. With our method with FIFO EF alone, the performance of the robot after training was better than with the DQL method and with the PER methods. In addition, our method did not affect the uniformed random sampling process of the experience replay mechanism, and the implementation was simpler compared that with the PER methods, with which we have to create a special heap structure to store experiences in the replay memory. Our method performed better than the DQL method especially when the replay memory was relatively small. With a smaller memory, it is difficult for the replay memory to maintain important experiences, and they also are forgotten quicker than with a larger replay memory. With our experience filtering mechanism, not only important experiences are kept longer, but also diversity of replay memory is also improved. These factors make the deep neural network learn more efficiently.The weakness of our method is with greedy EF. Instead of letting the agent forget the oldest memory, we let the agent forget the memory with highest similarity in the existing replay memory. This creates a bias when a neural network learns and causes overfitting on a set of fixed experiences. The FIFO EF does not have this problem; thus, it outperforms greedy EF.

8 CONCLUSION

We proposed a method of filtering and deciding whether to store a new experience in the replay memory to make the DQL algorithm work more effectively. Our method calculates the similarity (or distance) between

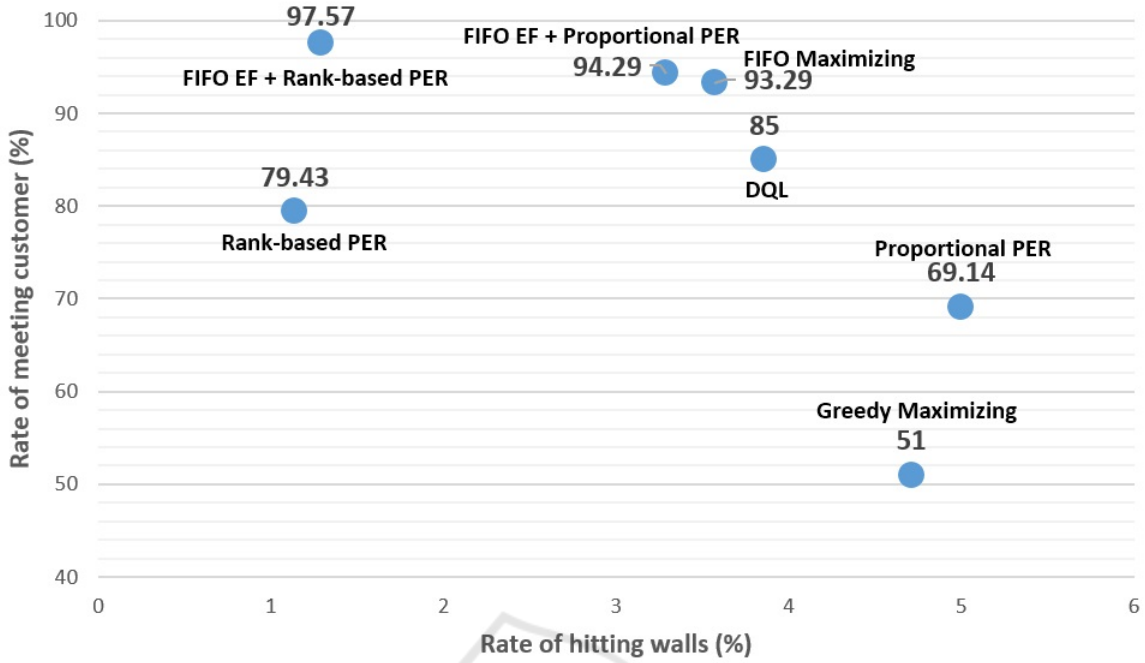

Figure 5: Robot's performance (capacity = 10^3).

a new experience and the rest of the existing experiences in the memory and converts the similarity (or distance) to a probability of storing this new experience. The higher the similarity (or the lower the distance), the lower the probability to store this new experience.

This method demonstrated superior performance to conventional methods when the replay memory has a limited size. Our method is compatible with the DQL and PER methods and further improves the performance of the DQL algorithm. In our experiment of training a robot to move in a virtual environment, the average reward it could receive was 18.66% higher than with the DQL method. We intend to apply our method to embedded devices and robots that have low-memory resources.

REFERENCES

Bernstein, S., Poznanski, R. R., Solomon, P., Wilson, S., Swift, R. J., and Salimi, B. (2001). Issn 0312-3685 subscription rates subscription rates (post free) for the 2001 volume of the mathematical scientist are:£ 11.00, us 18.15*or* a28. 60 all enquiries about the mathematical scientist, as well as other subscriptions, should be sent to the. *The Mathematical Scientist*, 26(2):136.

Fahimi, F. (2008). *Autonomous robots: modeling, path planning, and control*, volume 107. Springer Science & Business Media.

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*.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al. (2016). Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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.

Sadeghi, F. and Levine, S. (2016). (cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*.

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

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

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*.