Generative Replay in Deep Reinforcement Learning

Arflyno Chrisdion Pudayar Pahombar Ludjen
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
arflynochrisdionludjenr@student.utwente.nl

ABSTRACT

Deep reinforcement learning has been an improvement for reinforcement learning, in which it allows the learning of much more complicated environments. However, the issue of catastrophic forgetting is present in deep reinforcement learning. Catastrophic forgetting is a situation where the agent of deep reinforcement learning forgets past experiences. Experience replay is introduced following this issue, by allocating a buffer to store past experiences and to sample from them in the process of learning. A notable downside of experience replay lies in memory usage. Having the possibility of consuming a huge memory space, reduces the scalability of deep reinforcement learning. This research explores the alternative of experience replay in the form of generative replay. Generative adversarial network, a generative model, is implemented in combination with deep reinforcement learning to assess its performance in comparison to experience replay.

Keywords

Deep Reinforcement Learning, Generative Replay, Generative Adversarial Network, Experience Replay

1. INTRODUCTION

Reinforcement Learning (RL) is one of the leading fields in Artificial Intelligence. RL is a trial-and-error process to discover the actions that give the most reward by exploring their environment. States and actions, given by the framework of the Markov Decision Process (MDP), are recorded by the learner as an indication of the current situation of the learner at a given time [12]. Q-Learning, one example of RL, records state-action pairs in a form of a lookup table to which the learner can refer when needed[12, 1]. However, this proves to be a limitation to RL as most real-life problems can have an incredibly high state count, rendering it inefficient, especially in terms of memory, as the learner must record each of such states. This leads to a more sophisticated version of RL, Deep Reinforcement Learning.

Deep Reinforcement Learning (DRL) in its essence is a combination between RL and Deep Learning [1]. DRL utilizes neural networks for approximation of the state-action pairs. This allows DRL to work in more complicated environments compared to regular RL. An example of DRLframework is the Twin Delayed DDPG [3]. This will be the main DRL framework used in this research and more on this will be explained in the related works section. Neural networks, however, have a downside in which they tend to forget past knowledge and experience. This is referred to as Catastrophic Forgetting [2, 6]. In tackling this issue, Experience Replay was introduced.

The term Experience Replay (ER), firstly described byLin, 1992 [7], allows the learner of DRL to recall past experience. With this, the learner can from time to time use samples of past experiences stored through ER as input for its learning process. Again, however, similarly to the mentioned limitation of RL regarding memory, where ER may use a very large memory space as the environment becomes more complicated, thus reducing scalability. Studies were conducted on this issue in which the use of Generative Replay is suggested in a supervised learning environment [10].

At the time of this writing, there have been studies that researched the implementation of Generative Replay in machine learning. Generative Replay methods on supervised learning [10] and unsupervised learning [8] have all shown great potential in replacing ER. Furthermore, there has been very little research on the application of the Generative Replay method in DRL. One such study is the use of variational autoencoders in DRL [5]. Based on the previously explained issue, we will explore the possibility of Generative Replay in DRL, in particular the generative model of Generative Adversarial Networks (GAN).

The paper aims to answer the main question:

Main Research Question : How does GAN performs in comparison to ER in DRL?

In order to further understand the performance of GAN in DRL, the following sub-question is to be answered:

Sub-Question 1 Which aspects of GAN improve its performance in DRL?

This research results in showing the performance of implementing GAN in DRL, to find whether it is suitable in replacing ER. GAN algorithm is explored in order to test which aspects help improve its performance in combination with DRL. Through this research, it as well might give further insights into the exploration of this topic, especially for future research on this field.

2. BACKGROUND

2.1 Reinforcement Learning

Reinforcement Learning (RL) has been a staple to the Artificial Intelligence field. RL is learning what to do in

35^{th} Twente Student Conference on IT July 2^{nd}, 2021, Enschede, The Netherlands.

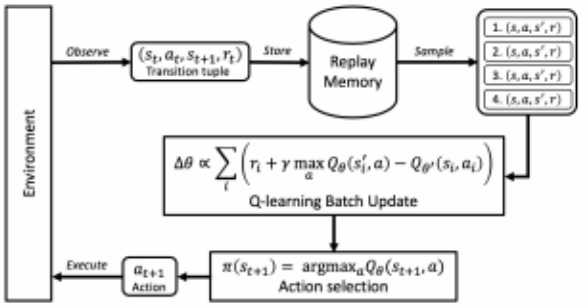

Figure 1. The process of Q-Learning [9]

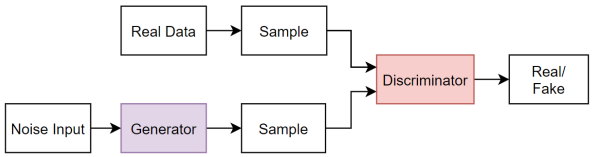

Figure 2. A representation of the GAN training scheme

order to maximize rewards. The agent (the performer of actions) is not told what actions to take but to explore the environment (an area of observation that is impacted by the agent) it is in and determining themselves the best course of actions [12].

Mathematically, RL is represented as follow. At any given timestep t, an agent situated in state $s \in S$ (S denotes the state space of the environment) is able to choose an action $a \in A$ (A denotes the action space of the environment) resulting in the agent arriving in the new state s' and receiving reward r. Through repeated exploration, the agent can then develop a policy $\pi : S \Rightarrow A$, in choosing action a given state s. A transition tuple formed as (s_t, a_t, s_{t+1}, r_t) acts as a representation the previously explained interaction of an agent in each timestep.

The essence of RL is to maximize the rewards gain as a result of taking an action a in state s [12]. One such algorithm of RL is Q-Learning. Figure 1 shows the process of Q-Learning. A state-action pair is utilized and stored in a lookup table in a form of $Q(s, a)$ mapped to the potential reward r to be received. Through such a method, the Q-Learning agent can determine which action is best to be taken regarding the state it currently is in. Though it is very effective, the lookup table can easily grow almost infinitely given a complex environment.

To address this issue, algorithms such as Deep Q-Network (DQN) were implemented. DQN utilizes Neural Networks (NN) [1] to approximate the state-action pairs, the $Q(s, a)$ of Q-Learning in this case, thus making a lookup table no longer necessary. As NN is applied, catastrophic forgetting [2, 6] the issue in which past experiences tend to be forgotten in comparison to the newer ones is present as well.

In addressing the issue of catastrophic forgetting, Experience Replay was introduced first by Lin et al [7]. It refers to the buffer that stores past experiences where it can be used to sample those past experiences in the effort to "refresh" the agent's memory.

3. RELATED WORKS

3.1 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed Deep Deterministic Policy Gradient (TD3) is the expansion of the original algorithm Deep Deterministic Policy Gradient (DDPG) by Silver et al [11]. DDPG is an actor-critic algorithm, in which the algorithm is built around two Q-Functions. The 'actor' determines the action be taken given a state, hence shaping the policy of the algorithm. Meanwhile, the 'critic' determines the value of such policy produced by the 'actor'. Introduced by Fujimoto et al [3], TD3 implemented three additional features to the original DDPG.

First, it applies two 'critic' functions instead of one and uses the smaller Q-values of the two functions to avoid overestimation in determining the best policy. Second, it applies a 'delayed' update to its policy resulting in a higher quality of policy updates as this makes sure that unchanged policies are not updated. Last, noise is added to estimate the policy, making the exploitation of Q-function errors much harder. TD3 is one of the best state-of-the-art algorithms[1] in the field of DRL and is one of the main reasons for applying it to this research.

3.2 Generative Replay

Generative Replay refers to the generation of new and relevant experiences based on past experiences, and it is becoming apparent in the field of Artificial Intelligence. Numerous researches are present on Generative Replay with examples on the application in continual learning [10], unsupervised learning [8] and DRL [5]. Applying Generative Replay to DRL proved to pose challenges as experiences to be generated by Generative Replay need to be perfectly fitted for DRL to receive positive outcomes. As such, exploration on the topic of Generative Replay in DRL is close to none leading to the reason for conducting this research.

3.3 Generative Adversarial Networks

Generative Adversarial Network (GAN) is a generative model introduced by Goodfellow et al [4], and is a generative model. GAN, in its essence, is a zero-sum game, consisting of a generator G and a discriminator D where the loss of one is the gain of another. The task of the discriminator D is to learn the distribution of data p_g and to determine such data to be the real data. Meanwhile, the generator G is tasked with generating data in hopes to fool the discriminator D into determining that the generated data is the real data. Both are trained simultaneously, where the generator G feeds its generated data to the discriminator D, resulting in G learning to generate data that greatly resembles the real data while D learning in better distinguishing real and generated data. The visual representation of the GAN framework can be seen in Figure 2.

4. METHODS

This section describes the contribution as well as the methods of experiment conducted in this research.

[1]Benchmark tests on different DRL algorithms can be found here `https://spinningup.openai.com/en/latest/spinningup/bench.html`.

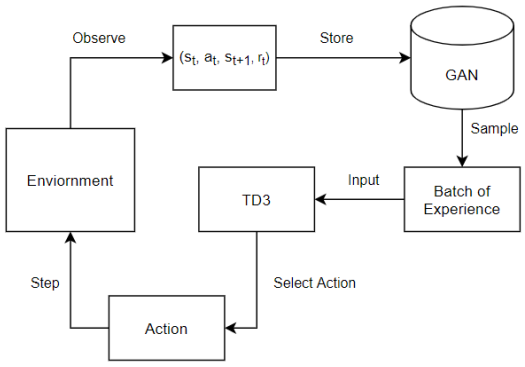

Figure 3. The process of TD3 learning with GAN

4.1 Combination of TD3 and GAN

The combination of both TD3 and GAN algorithm was done by replacing the replay memory in Figure 1 into that of the GAN algorithm. As the main difference, the new architecture consists of GAN receiving the transition tuple as training data and generating a new batch of experience using the generator of GAN. This new architecture is shown in Figure 3. The code can be accessed from `https://github.com/DionLudjen-13/GAN-TD3.git`

4.2 Explored Aspect of GAN

Alterations to the original codes were done in order to gauge and explore which aspects of the GAN affect its performance in DRL. There are a total of three methods of alteration conducted in this research. The results is discussed in Section 6.

4.2.1 Buffer Size for GAN

The main reasoning for applying Generative Replay on DRL is to avoid applying large-sized buffers such as Experience Replay. Thus it would be most interesting to explore whether a certain buffer size performs better than other sizes. Of course, only a small-sized buffer will be applied for GAN to be in line with the target of avoiding Experience Replay in DRL.

4.2.2 Sample Type

In order to learn, the discriminator of GAN requires 'real' experiences as a baseline in distinguishing the 'real' experiences and the 'fake' experiences produced by the generator of GAN. These 'real' experiences are retrieved from the buffer in two different methods: randomly chosen samples from the buffer, and the whole buffer as it is.

4.2.3 Buffer Type

Understanding that GAN observes and learns the distribution of the input it was given, the thought of implementing an additional buffer, the Optimal Buffer, besides the original one was considered. As the name implies, it refers to the experience buffer on the previous best episode. In total there are two metrics in determining an Optimal Buffer, an episode's reward and the final state of an episode that is closest to the finish state. The goal of this method is to explore whether specifying GAN in learning a specific buffer would produce better results.

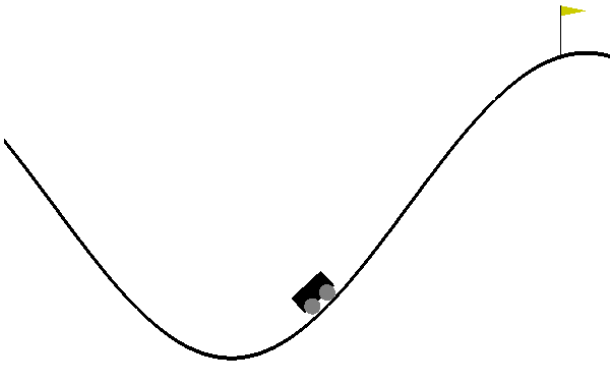

Figure 4. The Gym "MountainCarContinuous-v0" environment

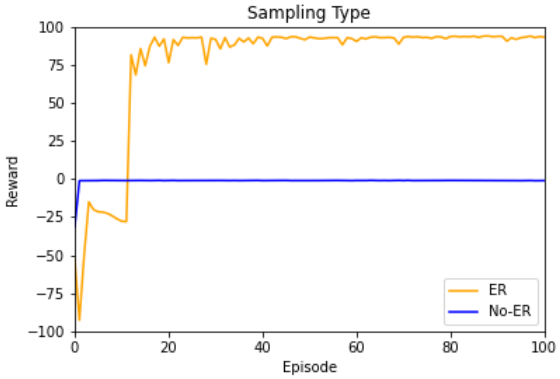

Figure 5. Experience Replay in comparison with limited buffer capacity

5. EXPERIMENTAL SETTINGS

5.1 Environment

The environment applied to the experiment is the "Mountain Car Continuous-v0" (Figure 4) of the Gym Classic Control Environment. The goal of the agent (the car) is to climb up a hill to reach the finish line. The agent observes the position its currently in as well as its velocity. Based on this observation the agent decides the acceleration it should take (continuous value in the range [-1,1], -1 indicates accelerating to the left and 1 accelerating to the right). The environment provides a challenge as the agent could not simply accelerate fully to the finish from its starting point, but rather gain momentum by accelerating backward followed by accelerating forward with that momentum. In addition, the reward system of the environment is more complicated in the sense that if an episode's reward is better than the previous, it might not necessarily mean that the agent performed better. To start with, the agent will be rewarded 100 points if it managed to reach the finish point where this is the only possible method of resulting in a positive reward accumulation of an episode. In each action taken, the agent will be deducted points determined by the number of movements the agent took. Simply said, if the agent decided to move a lot it will be deducted more points than if it decided to not move at all. Thus, based on this characteristic of the environment the reward accumulation per episode of both high negative and positive reward is valued more, although the latter is preferred the most. For every episode, the environment has a total reward range of [-100, 100].

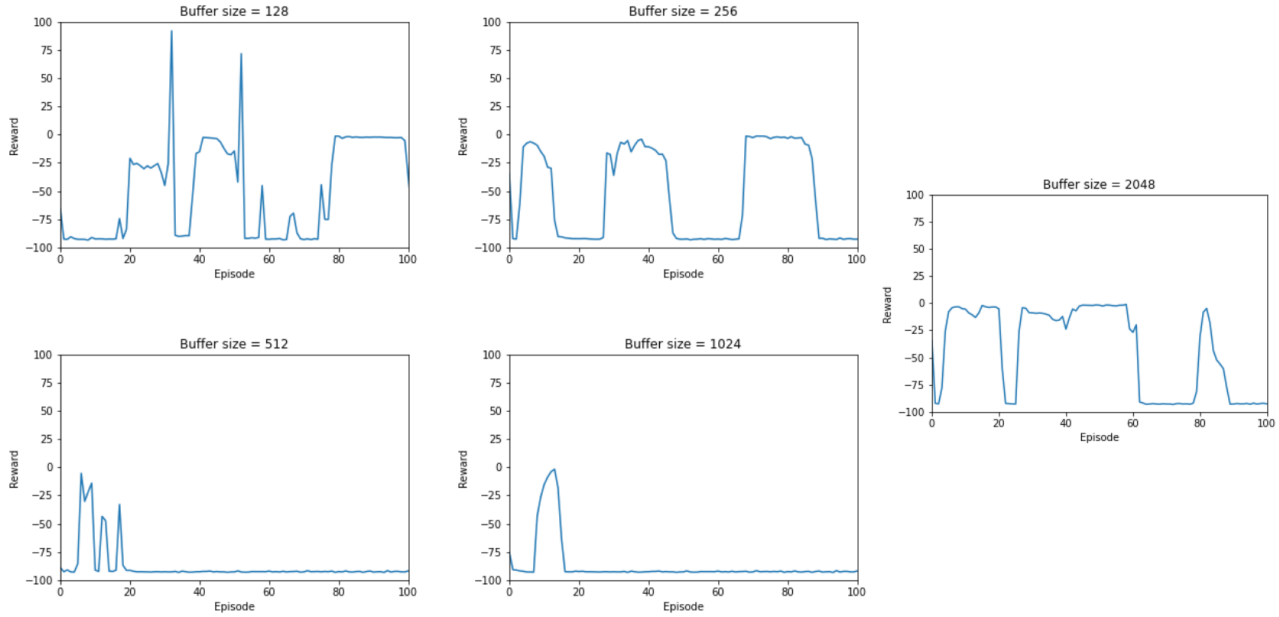

Figure 6. Buffer Size of GAN on sizes 128, 256, 512, 1024, 2048

5.2 GAN

The framework Pytorch was utilized for the implementation of GAN. This built-in neural network function of Pytorch allows to implementation of GAN in a simple manner. The GAN observe an experience tuple of 7 features both for the Generator and the Discriminator model. These features are determined by the information observed by the environment "MountainCarContinuous-v0", consisting in order of two features for states, one for both actions, two for the next states, one for the reward collection, lastly followed by a boolean if the episode is completed. The representation of the experience tuple are as follow where s and ns denotes state and next state respectively:

$$[s_0, s_1, action, ns_0, ns_1, reward, done] \quad (1)$$

Definition 1. The visual representation of the experience tuple states mass equivalence relationship.

5.3 TD3

The original implementation of the TD3 was provided by Fujimoto et al [3] in their GitHub page[2]. In order to train the TD3 algorithm, a replay buffer is necessary, in which the replay buffers applied will be Experience Replay, a regular buffer with limited capacity (a total of 256 experience tuple is stored at a time), and lastly the implementation of GAN. Initially, the performance of Experience Replay and a regular limited buffer will be compared to gauge the effectiveness of Experience Replay and is shown in figure 5. This figure describes that Experience Replay performs significantly better than with a limited buffer.

6. RESULTS

This section describes the experiment result of the previously mentioned method of experiments. In order to make all the experiments feasible, different alterations and methods used on the GAN algorithm applied the baseline parameters. Describing those parameters specifically, the baseline size for the buffer of GAN is 256, the sampling type is the whole buffer in the training process, and lastly, for the buffer type, the original buffer is used. In all of the experiments, the episode limit was 100 episodes per experiment.

6.1 Buffer Size of GAN

The result on this method of experiment is given in 6. The buffer size of GAN explored are on size 128, 256, 512, 1024, and 2048. It would seem that the buffer size of GAN did not give any significant improvement on the performance of GAN in DRL. Buffer size of 128 showed a promising result as it manages to reach a positive reward but then followed by a high negative reward. As a reminder due to the reward system of the environment, high negative reward implies that the agency conducted a lot of movement. The steep fall from positive reward to high negative reward indicates that the agent tried to climb up the hill but did not manage to reach the finish line. The result seemed apparent as well in the buffer size of 256 and 2048 showing that the agent does regress after a series of the episode trying to climb up the hill, indicated by the high negative reward, with the difference being no positive reward were achieved. Interestingly enough, the buffer size of 512 and 1024 showed that the agent made considerable movement continuously without regression but still cannot reach the finish line.

6.2 Sampling Type

The experiment on this alteration produced interesting results. Figure 7 shows that both sampling types proved to at least allows the GAN to provide samples that in return convince the agent that moving is better rather than staying still, although given that moving might result in worse rewards. Out of the times, this alteration is conducted, this particular result on both random sampling and the whole buffer sampling shows the most promising outcome. Random sampling over the times almost always shaping the TD3 policy's into choosing that moving is better, but regretfully still cannot reach the finish line. Meanwhile, with whole buffer sampling, the agent managed to reach

[2] `https://github.com/sfujim/TD3`

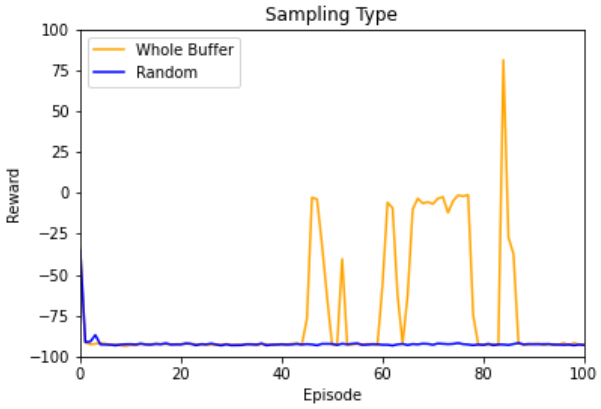

Figure 7. Performance of Random sampling and Whole Buffer sampling

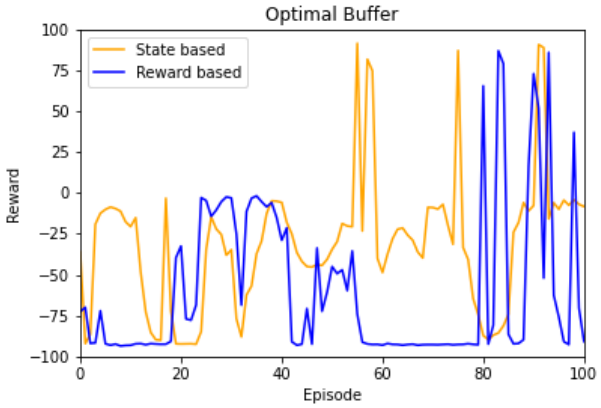

Figure 8. Performance of Optimal Buffer

the finish line once followed by a regression of constantly moving without being able to finish. Based on this, although random sampling shaped the TD3 policies to that of always choosing to move, whole buffer sampling gives the outcome as it received a positive reward.

6.3 Buffer Type

As previously described, the Optimal Buffer applies two types of parameters in deciding which buffer is more optimal, by evaluating the state of the agent in each episode and by evaluating the rewards per episode. This evaluation is done inline as the agent explores the environment, and each Optimal Buffer is replaced by a better one if any is found to be present. In regard to the condition of the reward system, a constraint was added. Through observation, the agent showed considerable movements if the resulting reward of the episode is either positive reward or in the range [-100, -10] for negative reward.

After multiple runs on the types of parameters, very surprisingly both types of parameters performs incredibly well in one of their instances, more so than the original buffer used in the previous experiments and is shown in Figure 8. One such reasoning for this result is perhaps due to the fact that GAN learns the distribution pattern of the buffer, and if such buffer is the result of the most optimal episode (the Optimal Buffer), GAN would learn to generate batches of experience that resembles the buffer. Even if previously the Optimal Buffer does not achieve positive rewards it is possible for GAN to generate some experiences that can cover what the previous Optimal Buffer could not do.

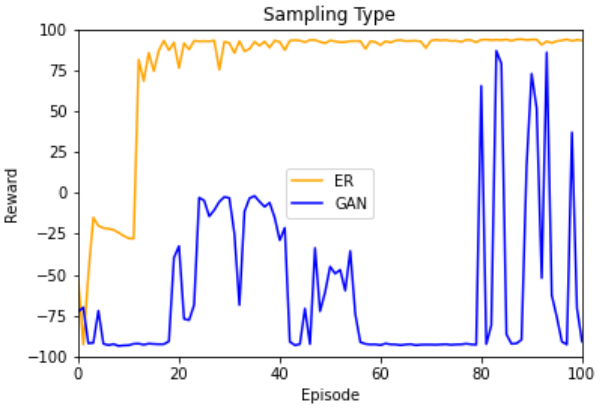

Figure 9. Performance comparison between Experience Replay and GAN in TD3

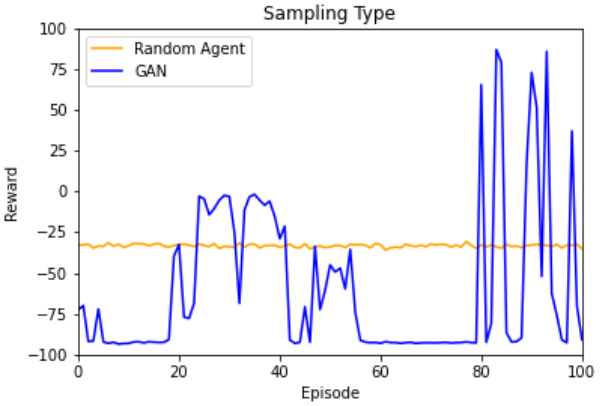

Figure 10. Performance comparison between Random Agent and GAN in TD3

6.4 Comparison between Experience Replay and GAN

Figure 9 shows the direct comparison on the performance of the agent in the environment. It can be seen that Experience Replay still managed to perform considerably better than GAN. Experience Replay manages consistently, from the first instance of receiving a positive reward, to keep achieving positive reward without any regression whatsoever. Though the result of this comparison is heavily on the side of Experience Replay, at around episodes 80-100 it shows that GAN really does have potential in rivaling Experience Replay, in particular with the implementation of Optimal Buffer. Perhaps if the regression happening after each positive reward, which is likely caused by the agent missing that last couple of steps from the finish line, could be reduced by introducing new parameters and alteration GAN can prove to be promising in replacing Experience Replay. In addition, GAN still performs better than a random agent as shown in Figure 10. GAN has the capabilities of actually achieving positive rewards while a random agent produced some movements but not to the extent of reaching the finish line.

7. CONCLUSION

7.1 General Conclusion

In this research, GAN is implemented in place of Experience Replay and a total of 3 aspect exploration of GAN were introduced to the algorithm to gauge the effectiveness of GAN in DRL. Answering the sub-question, the results show that improvement can be seen considerably in replacing the buffer type used for training the GAN from the original buffer to the Optimal Buffer. Altering the buffer size of GAN as well as sampling type does improve the GAN but not to the level that replacing the buffer type manages to do. Onward to answering the main research question, regretfully GAN did not perform well in comparison to Experience Replay. Experience Replay manages to achieve positive rewards and never regress, in which the GAN was not able to do the same. Overall, this research shows that in comparison to ER, the performance of GAN still falls short. However, GAN did show that Generative Replay has potential in its implementation in combination with DRL, especially as a replacement for ER in GAN.

7.2 Weaknesses and Future Works

A weakness of this research lies in the computational capabilities of the computer used in this experiment. Overall, the episode was capped to 100 episodes as it took around 45 minutes in order to run each one of the experiment instances. Due to this episode limitation, the quality of the observation is reduced as a few instances of the experiment began to show promising results close the 100 episode count limit (an example is on the performance of Optimal Buffer in Figure 8). Another weakness of this research is the nature of GAN in comparison to the environment used in the DRL. In nature, GAN observes and learns a distribution, which then results in the capability of producing data differently for each batch but with the same distribution. It seems that this does not fit perfectly well in the environment "MountainCarContinuous-v0" used in this research as although GAN managed to learn the distribution of the buffer of experience, the generated data of that distribution might not necessarily be the best course of action in order to achieve the maximum possible reward.

In regards to the weakness of the environment, this leads to a possible future work of exploring a different environment that considers data distribution as an action to be more important rather than an environment that only considers action on singular data points for each timestep. Another possible future work on this research is the incorporation of GAN learning in TD3. In this research, both GAN and TD3 learn independently. GAN learns as its buffer is filled by experiences while TD3 learns as it samples from the generation result of GAN. As TD3 learns on creating the best policy for the agent, incorporating the results of GAN learning in TD3 learning might further improve the generator of GAN, in a sense that the newly generated data would be more fitting to the policy of TD3.

8. REFERENCES

[1] V. F-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018.

[2] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3:128–135, 1999.

[3] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. 2018.

[4] I. J. Goodfellow, J. P-Abadie, M. Mirza, B. Xu, D. W-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. 2014.

[5] B. Imre. An investigation of generative replay in deep reinforcement learning. 2021.

[6] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, D. K. C. Clopath, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 3:128–135, 2017.

[7] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:342–351, 1992.

[8] D. C. Mocanu, M. T. Vega, E. Eaton, P. Stone, and A. Liotta. Online contrastive divergence with generative replay: Experience replay without storing data. *CoRR*, abs/1610.05555, 2016.

[9] A. Raghavan, J. Hostetler, and S. Chai. Generative memory for lifelong reinforcement learning. *CoRR*, abs/1902.08349, 2019.

[10] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. *CoRR*, abs/1705.08690, 2017.

[11] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. 2014.

[12] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.