An Investigation of Generative Replay in Deep Reinforcement Learning

Barış İmre
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
b.imre@student.utwente.nl

ABSTRACT

Catastrophic forgetting is an issue that persists in all deep reinforcement learning settings. Traditionally, catastrophic forgetting is solved by using experience replay, which is a buffer that remembers past experiences and uses these experiences to train an agent. This solution is effective in overcoming catastrophic forgetting but it heavily depends on memory. It would be a big breakthrough, if catastrophic forgetting could be solved without using memory, since this would dramatically increase scalability. To this end, a generative replacement is proposed in this paper for experience replay. Variational autoencoders are tested as a generative model on a simulation setup with a state of the art deep reinforcement learning algorithm. This method is called generative replay. Multiple methods of training and interaction are tested in order to explore the combination of generative replay and a deep reinforcement learning agent. Even though this paper does not present overwhelmingly positive results, it gives many insights on combining these networks in an experimental setup, and explores future possibilities for this approach.

Keywords

Deep reinforcement learning, catastrophic forgetting, variational autoencoders, generative replay

1. INTRODUCTION

Deep reinforcement learning (DRL) has great potential when it comes to addressing real life problems. However, there are of course still issues that need to be addressed. One common problem in many deep learning tasks is catastrophic forgetting [12], where the agent prioritizes newly learned information over past experiences and essentially forgets what it already learnt in the past. This is even more common in DRL settings since the learning happens in a sequential fashion. One of the most common ways of overcoming catastrophic forgetting is to remember past experiences and reuse these experiences to firm the learnt knowledge. Almost all of these methods require memory to store the past, reducing scalability. For example, a simple humanoid model such as the OpenAI Gym [1] environment "Humanoid-v2", can take up to 3 GBs of memory for a single humanoid. Scaling this to a crowd would already surpass the memory capabilities of a high end mainstream computer. This is a big issue in DRL, if real life problems are ever to be solved using these methods. For this reason, in this paper, we propose an approach to discover if it is possible to remember past experiences in a meaningful way without relying on memory to overcome catastrophic forgetting. We investigate the feasibility and performance of a generative replay in deep reinforcement learning.

As explained in the related work section, research in this area is almost nonexistent. Due to the lack of already existing methods, a path must be carved for methods that are promising. The purpose of this research is therefore twofold. One, this work aims to investigate how well a variational autoencoder can replace experience replay (memory) in DRL with a few different methods. Two, this paper aims to identify promising future methods and models as well as identify the weaknesses of the proposed methods. This way research can be picked up where this paper ends and have the hindsight knowledge of this paper. Findings to this end are discussed in depth in the section future work and concrete open questions for future research are formulated. The paper aims to answer the following research questions:

Research Question 1: To what extent are variational autoencoders suitable to replace experience replay in DRL?

Research Question 2: What are some methods that improve or hinder the performance of variational autoencoders in DRL as a generative component?

Research Question 3: What are the most promising paths to take in order to replace experience replay in DRL with a generative method?

The structure of the paper is as follows. First, the basics of all the components are explained and background knowledge is given. Then the experimental setup is explained and then the models and the various methods are laid out and tested. The results of these tests are discussed and explained in detail. Lastly a discussion is given for the many reasons why this approach did not produce stellar results and what can be done to improve on this work.

2. BACKGROUND

Reinforcement learning (RL) is the process of learning to maximize a numerical reward signal, meaning it is learning to map certain situations to certain actions based on a reward system. The agent learning, must discover which actions yield which rewards in certain situations and must learn from these experiences [17]. Reinforcement learning methods are especially effective in sequential decision processes, formalized as Markov decision processes (MDPs). We can mathematically formulate this process as follows.

34th Twente Student Conference on IT Jan. 29th, 2021, Enschede, The Netherlands.

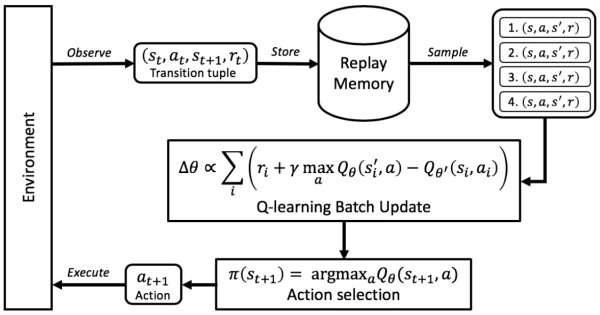

Figure 1. Reinforcement learning with experience replay [15]

At any given time step t and a state $s \in S$, an action $a \in A$ can be chosen based on a policy learnt $\pi : S \rightarrow A$. This will produce a reward r and transition the environment to a new state s'.

A RL algorithm most commonly aims to maximize the expected return in the long run, by finding an optimum policy π_*. A transition tuple can be formalized per time step $t \geq 0$ as (s_t, a_t, r_t, s_{t+1}) , where s_{t+1} is the next state the environment transitions into because of the action a_t, with the reward r_t. A common RL algorithm then, would try to maximize the expected return for all state action pairs $(s, a) \in S \times A$ [2].

One very common solution, *Q-learning* [18], achieves this by simply storing estimations of all state action pairs mapped to rewards, using a Q-table, as commonly referred. As a transition happens, the value $Q(s_t, a_t)$ corresponding to the state action pair at time step t gets updated with the target $r_t + \gamma max_{a \in A} Q(s_{t+1}, a)$ for each $t \geq 0$. Important to note, γ denotes the discount rate parameter, which is how much the learning agent cares about the future states and rewards compared to the immediate reward.

A clear downside of this basic Q-learning algorithm is of course its use of tables to store data, making it extremely hard to scale. More recently, researchers have combined neural networks with the idea of Q-learning to produce more scalable and modern Deep Q-Networks (DQN) [13], taking advantage of neural network function approximation. Practically, approximating the tables in a Q-learning setup using neural networks. This is the point where catastrophic forgetting starts becoming an issue, since neural networks tend to discard old knowledge in favor of newer knowledge [7].

A clear way of overcoming this issue is *experience replay* (ER) [11]. Where usually a buffer is kept of older experience tuples and at selected intervals a sample from that buffer is used to "remind" the agent of past experiences. This is a widely used and extremely effective method when combined with suitable reinforcement learning algorithms, however the obvious downside is again the memory usage. Figure 1 shows a generic architecture where experience replay is utilized with a replay memory component.

3. RELATED WORK

This section explains many of the concepts used in this paper together with other research that closely relates to this paper.

3.1 Variational Autoencoder

3.1.1 Dimensionality Reduction

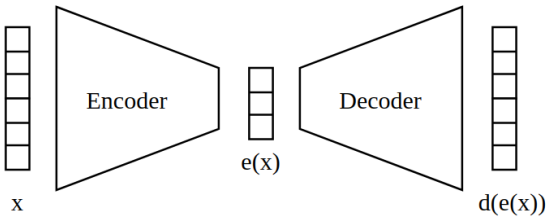

Figure 2. Overview of an autoencoder with data x encoded and then decoded, $x \approx d(e(x))$

Dimensionality reduction is the concept of taking high dimensional data and abstracting this data to a lower dimensional representation. This compression should keep the meaning behind the data, and ideally be reversible in some way.

3.1.2 Autoencoder

An autoencoder [8] is an artificial neural network that consists of two parts; an encoder and a decoder. The encoder part is a neural network that compresses the input features to a bottleneck, and the decoder part reverses this process. With some initial data x, an autoencoder can compress the feature set x using the encoder network to $e(x)$ with a smaller dimension. Then using the decoder network the initial x can be fully or partially reconstructed as $d(e(x))$. Figure 2 shows the architect of an autoencoder with the input and output. Feature set x is bottlenecked into $e(x)$. If a perfect autoencoder is used, x is equal to $d(e(x))$. However in an imperfect setting the difference between x and $d(e(x))$ represents the loss this network has. Hence, an autoencoder is trained on a dataset in an iterative way by running all data points through the network, calculating the loss between the original and the reconstructed, and optimising the parameters based on this loss.

Definition 1. The loss equation of an autoencoder

$$loss = ||x - d(e(x))||^2$$

3.1.3 Latent Space

Diving deeper into the distribution to the compressed version of a dataset, one can usually observe a grouping behaviour within certain data points. The latent space, is the representation of this compressed dataset where similar data points group together. Later in this paper the latent representation of the experiment setup is discussed in detail.

3.1.4 Generation of New Data

The objective of the generative model in this research is to generate meaningful and varied new data. A way of generating data is using the latent representation and the decoder part of an autoencoder. If some noise is given to the decoder in the shape of the latent variables, the decoder might produce data that would look like real data points from the dataset this network is trained on. An improved version of this method then is to sample from the latent representation rather than to use noise, assuring that the distribution of the original data is followed.

3.1.5 Variational Autoencoder

The reason for the variational autoencoder (VAE) rises from an issue with autoencoders. Even though, autoencoders are great at compression and decompression, they

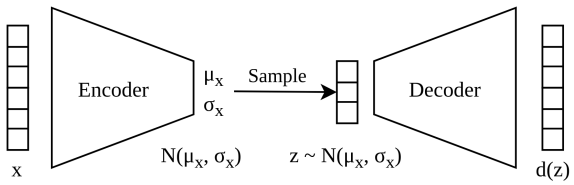

Figure 3. Diagram of a VAE with input x, mean μ_x ,covariance σ_x , latent sample z from a normal distribution, and output $x \approx d(z)$

do not account for the regularity of the latent space in the encoder network. They are simply trained by looking at the input and the output of the network. This makes sampling from the latent space and generating new data hard and ineffective using autoencoders.

Variational autoencoders introduce a middle step between the encoder and the decoder networks, in order to regularise the latent space and representation of inputs. This can ensure that the latent space has properties that enable a generative process. In practical terms, VAEs encode a data point as a distribution over the latent space instead of a fixed representation. This is usually formalized in terms of the means and the covariances of the latent distribution. Then a latent point is sampled from these distributions and used as input for the decoder. Figure 3 shows the architecture of a simple VAE.

VAEs use Kullback–Leibler divergence [9] (KL-divergence) as a measure of regularity in the latent space with the mean and covariance terms. This is represented in the loss equation for a VAE, modifying the loss function of an autoencoder. Another term C is added as a coefficient to modify the significance of this regularization. More discussion about the loss functions used in the implementation and theoretical VAEs is give in the methodology section.

Definition 2. The loss equation of a variational autoencoder, KL refers to Kullback–Leibler divergence.

$$loss = ||x - d(z)||^2 + C * KL[N(\mu_x, \sigma_x), N(0, 1)]$$

3.2 State of the Art in DRL

3.2.1 Twin Delayed Deep Deterministic Policy Gradient

The idea proposed by Fujimoto et al. [3], Twin Delayed Deep Deterministic policy gradient algorithm (TD3), is a state of the art deep reinforcement learning algorithm. It is built on Double Q-learning [5] and Deep Deterministic Policy Gradient (DDPG) [4] algorithms. TD3 has been tested repeatedly for stability, efficiency, and compatibility for the environment we plan to use. TD3, without its many improvements upon the modern algorithms, still uses the very effective replay buffer to overcome catastrophic forgetting.

TD3 is an actor-critic algorithm, meaning that it uses a neural network to estimate a value function "critic" and another neural network to update the policy distribution "actor" [19]. This is also the case for DDPG. On top of DDPG, TD3 builds three innovations. One, TD3 learns from Double Q-learning and uses two critic networks instead of one. It elects the lowest value of the two critic networks when making an estimation reducing bias and overestimation. Two, TD3 uses delayed updates to its policy network in order to stabilize the value network quickly. Three, TD3 adds clipped noise to a selected action in order to estimate a target. All these three improvements make it an optimal deep reinforcement learning algorithm, hence why we use TD3 as the DRL component of this research.

3.2.2 Experience Replay

First described by Lin et al [10], experience replay in reinforcement learning has been the way for the past decades. In the paper by Fedus et al. [2], a detailed investigation on experience replay can be found, showing the effects of replay capacity (size of the replay buffer) and the replay ratio (ratio of learning updates to experience collected). The paper does not directly relate to our work since what we propose is to remove ER, however, the insights gained in this paper could be crucial to the performance of our implementation.

3.3 Generative Replay

Generative replay [16] is an idea that is becoming common in the machine learning community. Inspired from the human brain, generative replay refers to the approximate generation of relevant experiences to a situation based on previous knowledge. Experiments have been done replacing experience replay with generative replay in continual learning [16] and in unsupervised learning [14] with mostly successful results. However, progress in the reinforcement learning area with generative replay is almost non-existent. One paper has been published by Raghavan et al. [15], explaining the abstract idea but the paper does not present any results or share details of implementation.

4. METHODS

This section of the paper describes the methods that will be used in this research. These methods are alterations that will be done to the basic system described in subsection 5.4. The experimental results of these methods are given in section 6.

4.1 Start Delay

In DRL a very common practice is to delay the training of the agent in the beginning of an experiment to collect some data from the environment. This normally supplies enough data to the replay memory to effectively start the training. The same approach can be taken in generative replay to let the generative model develop an understanding of the environment.

4.2 Normalization in the Latent Space

As explained before the loss equation of a VAE can be expressed as the difference between input and outputs in combination with the KL-divergence of the latent space with a normal distribution. The goal of the first term in this equation is to match the inputs and reconstructions as close as possible. The KL-divergence on the other hand, forces the latent space to resemble more of a normal distribution. Experiments will be conducted to see if the affects of different levels of KL-divergence in generative replay.

Definition 3. Loss function of the VAE in practical terms

$$loss = binary\_cross\_entropy(x, d(z)) + C * kl\_divergence(N(\mu_x, \sigma_x), N(0, 1))$$

4.3 Buffer for VAE

The process of training the VAE as information comes can also cause catastrophic forgetting. One way to cope with

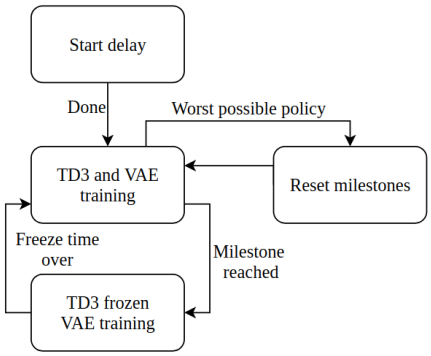

Figure 4. Delayed TD3 training architecture.

catastrophic forgetting in the VAE might be to keep a small buffer of experiences as they are collected. Once a certain amount is collected the VAE can be updated with a batch update of experiences. In the spirit of keeping the memory usage to a minimum, this buffer will not be a big value, such as the ones used in experience replay.

4.4 Delayed Training

With the hindsight knowledge of the previous methods, a more complex architecture for training the DRL agent is discussed. The idea behind this architecture is to freeze the training of TD3 after a sizable increase in the rewards is collected. The purpose of this is to give the VAE some time to adjust to the new experiences that are being seen from the environment. Essentially, using a peak TD3 version to discover the environment to generate more relevant experiences. This method aims to achieve a gradual and sustained increase in the rewards collected during training. Detecting a good enough policy is done by using milestones. Milestones in this context can be defined as predetermined reward values that represent a gradual but decisive increase. If a moving average of the previous rewards surpasses a certain milestone, TD3 is frozen. These milestones are implemented as an array of values and as one is surpassed it is removed from the array. This is intended to chase gradual increase. In practice, it is also necessary to implement a return to the original milestones if the policy degrades too much. Figure 4 outlines the architecture of this delayed training method as a state diagram.

5. IMPLEMENTATION

In order to answer the research questions stated in the introduction, we created a python module using pytorch, cuda, and numpy. For the environment where the experiments took place the OpenAI Gym [1] environemnt "InvertedPendulum-v2" is used. The Adam optimizer was used for updating the network parameters[6] in all components of the system.

5.1 Environment

The inverted pendulum is a classic control task with a pole on a cart. The objective is to balance the pole on top of the cart as long as possible and the pole can be controlled by moving the cart with certain velocities. Agents try to learn a policy which tells what velocity should be applied to the cart based in the state of the pole. For every time step that the pole is standing, one point of reward is given. This is capped at 1000 and the episode ends if the cap is reached. Figure 5 shows a screenshot of the environment

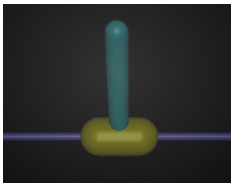

Figure 5. InvertedPendulum-v2 environment

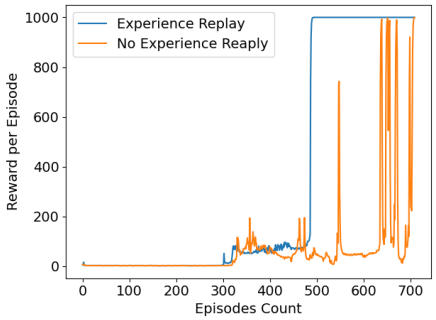

Figure 6. InvertedPendulum-v2 environment training progress, the difference of experience replay

where all the experiments are made.

5.2 VAE

A simple variational autoencoder was implemented from scratch with a feature size of 11 and a latent size of 3. The feature size is derived from the length of one experience in the environment used. InvertedPendulum-v2 provides 4 floating point numbers for a state representation, one for an action, one for the reward collected in the experience, and a boolean if the episode is over. The following tuple represents an experience with s for state and ns for new state.

Definition 4. The form of an experience tuple in the environment

$$[s_0, s_1, s_2, s_3, action, ns_0, ns_1, ns_2, ns_3, reward, done]$$

Of course, all values are scaled down to the interval $[0, 1]$ for the neural networks. The combination of binary cross-entropy and KL-divergence are used as the loss function for the VAE. The ratio of the two function in the loss equation is discussed later on this paper.

5.3 TD3

Fujimoto et al. [3] has already provided their work as an open source python module. The TD3 implementation used in this paper is adapted from that module without changing any major attribute. The experience replay component is replaced with the VAE and some helper python functions for the experiments. To understands the baseline of TD3 a small experiment is made to compare the effect of experience replay in reinforcement learning. Figure 6 shows the difference experience replay achieve in training speed and stability in the experiment setup described above. The oscillating behaviour of the agent without ER is consistent with catastrophic failure as is the regression even if a good policy is found. Observe that with experience replay, the agent almost never regresses.

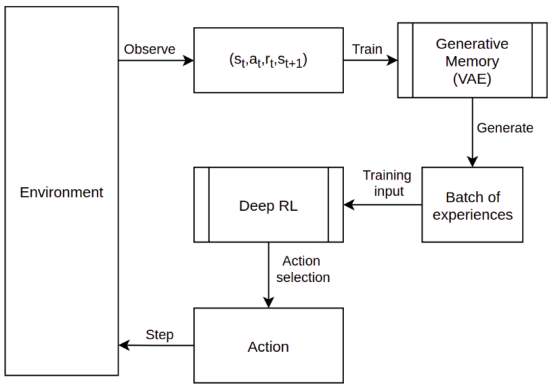

Figure 7. Initial architecture with VAE generating experiences and training as new experiences come

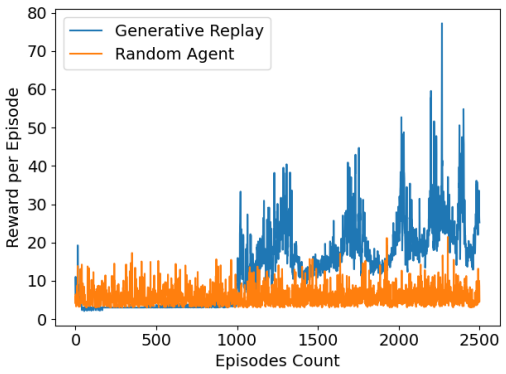

Figure 8. Generative replay against random agent

5.4 Initial Architecture

Figure 7 shows the architecture of the initial design. This architecture simply replaces the experience replay in regular DRL with generative replay by replacing the replay memory in Figure 1 with the proposed generative model, variational autoencoder. The sampling process is then changed to generation of new data using the VAE instead of random data selection from a buffer. All the code for the paper can be found at `https://github.com/Barisimre/TD3-Generative`.

6. EXPERIMENTS AND RESULTS

This section of the paper describes the various experiments conducted. In all the experiments, all other factors are kept on the same baseline as much as possible with only altering the independent variables. The following sections will present alterations and possible improvements to the various components in this study. However, these changes would not carry any meaning without first observing the proposed idea in its simplest form.

6.1 Initial Experiment

Initially, generative replay is tested without altering any aspects of the design. This experiment and every experiment of this paper presents data that is usually collected from training scenarios. This data is consistent of rewards collected over a certain amount of episodes. Looking at the reward values compared to a random agent in the environment, it is clear that the agent with generative replay manages to learn something about the task. Figure 8 shows this comparison clearly as the rewards are much higher on average and generative replay has a slight upward trend. Note that with or without experience replay the agents achieve considerably higher rewards compared to generative replay. This is to be expected since without any modification generative replay is still much inferior to basic reinforcement learning without experience replay. In Figure 6, the reward values rise as high as the environment allows which is 1000 whereas with generative replay this number never surpasses 100. The following sections explore possible improvements to generative replay.

6.2 Start Delay

To investigate the effects of start delay three different delays are tested. Figure 9 shows the rewards collected while training using the basic generative replay setup with different values of start delay. Beyond a delay of 15000 start steps, no improvement was observed. Moreover, the role of chance seems to be almost as significant as the start delay used as the experiment with the smallest start delay has the best maximum reward. We can conclude that all the given start delays are appropriate for future experimentation and that there seems to be no visible benefit of a bigger start delay.

6.3 Normalization in the Latent Space

First, as shown in Figure 10, without KL-divergence the latent space is tightly organized into groups. This is logical since this experiment is done with randomly sampled data from the environment, meaning there are two distinct cases. The smaller group represents the last experiences of episodes and the bigger group the rest. As KL-divergence becomes more prominent this grouping disappears. This is helpful for generating new data since sampling from the latent space becomes more accurate. Also, as with increasing levels of KL-divergence in the loss function, the latent space gets scaled down to an interval closer to $[-1, 1]$.

Second, as KL-divergence forces the latent space to be more normal, the actual output values get more of the "averaged out" effect. Every feature in the reconstructions skews closer to the average of that feature over the dataset with more KL-divergence. This is not particularly easy to represent with a graph however Table 1 gives example experience tuples with various reconstructions. The affect of this behaviour in the DRL with increasing KL-divergence setup as one can guess is quite negative. Since the environment presents a control task, loosing resolution on the state numbers hurts the training process. This becomes obvious even with the smallest coefficient C.

Finally, looking at the affect of C on the training setup mentioned above, we can conclude that any loss in the detail of states or other parameters due to normalization reduces the total and average rewards gained. As shown in Figure 11 even with $C = 0.001$ training is deeply worsened compared to no KL-divergence and any value higher than 0.001 shows almost no learning for the agent as seen from the reward values achieved with $C = 0.01$. To conclude, adding normalization to the latent space is an intuitional and logical idea, however due to the inaccuracies in the data generated and the effects of this in DRL performance, normalization is not used in this paper beyond this point.

6.4 Buffer for VAE

So far the problem seems to have been the stability issues caused by the two different models training at the same time. And even though it is not a solution to that, keeping a small buffer of experiences before training the VAE might increase stability. To be perfectly clear, in all

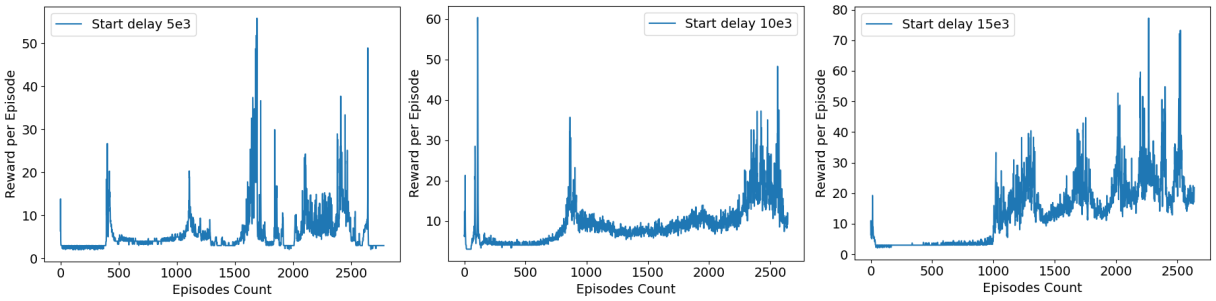

Figure 9. TD3 training rewards collected using the basic VAE setup with different start delays

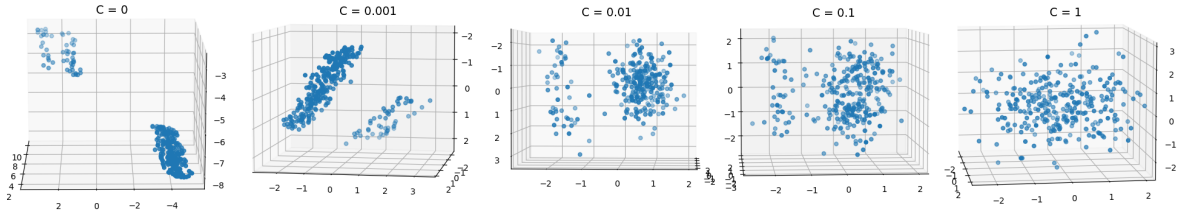

Figure 10. Latent space distributions with the coefficient C increasing from left to right. The effect of normalizing can be observed clearly. Data acquired by training identical VAEs with 100,000 experiences sampled from the environment. Images show the latent representations of 300 samples from each model with different coefficients.

Table 1. An example of reconstructions with various KL-divergence levels. The tuples are experiences in the form outlined in subsection 5.2

Coefficient C	Experience
Original	0.503, 0.494, 0.577, 0.325, 0.500, 0.506, 0.487, 0.576, 0.330, 0.050, 1.000
0	0.503, 0.494, 0.578, 0.329, 0.510, 0.506, 0.487, 0.575, 0.329, 0.050, 1.000
0.001	0.504, 0.493, 0.576, 0.331, 0.511, 0.506, 0.486, 0.576, 0.327, 0.050, 1.000
0.01	0.500, 0.501, 0.498, 0.504, 0.529, 0.500, 0.499, 0.501, 0.498, 0.049, 1.000
0.1	0.501, 0.500, 0.496, 0.506, 0.514, 0.500, 0.500, 0.497, 0.511, 0.050, 1.000
1	0.500, 0.499, 0.501, 0.498, 0.499, 0.501, 0.500, 0.502, 0.496, 0.050, 0.166

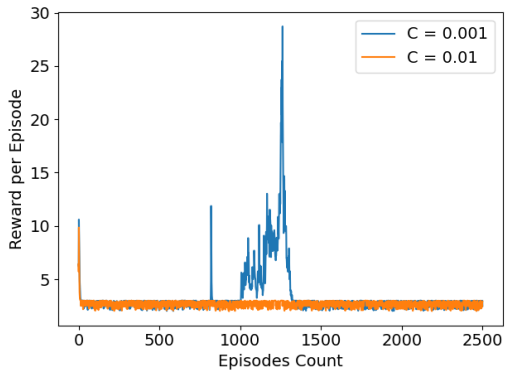

Figure 11. Training rewards with different KL-divergence coefficients

the previous experiments the VAE did have a buffer of 64 experiences simply because of practical reasons. With modern machine learning frameworks it is much faster and efficient to perform batch updates to neural networks and 64 is the size of a normal batch. The following experiments are done by increasing the buffer size by using multiples of two as multipliers with the batch size. These experiments is done for all powers of two until 64 meaning an effective range of buffer sizes from 128 to 4096. In practical terms, the VAE is not trained until the buffer is filled with experiences and once the buffer is full it is trained with all the experiences in the buffer. The buffer is then emptied.

Looking at the training sessions in Figure 12 one can observe that increasing the buffer size does not add any extra improvements to the system. In fact, as the buffer size increases it seems that the overall learning ability of the DRL algorithm decreases in general. The role of a lucky run is still present here. The decrease can be attributed to a variety of reasons. One of them might be that by increasing the buffer size, the intervals in which the VAE is updated decreases. This increase in the update intervals might make the generated experiences less recent and reliable. Another more likely explanation is however, that even without the updates on the VAE, TD3 gradually worsens after a peak performance as learning continues. In any of the peaks that one can observe in the figures, there are visible peaks of good policy by TD3, followed by a decline to almost the worst policy. This might occur since with a better policy, experiences that have never occurred in the system before are discovered. The VAE has not seen any of these experiences before and if it is not immediately updated, it will not sample such experiences. Meaning, even if TD3 develops a good policy, it cannot be reinforced since VAE cannot generate data that reflects these new experiences. In Figure 13 it is more apparent that right after the VAE updates, there are a relatively good policies from TD3 followed by a regressions to bad policies. In short, generated data from the VAE seems to lack in freshness as TD3 improves the policy.

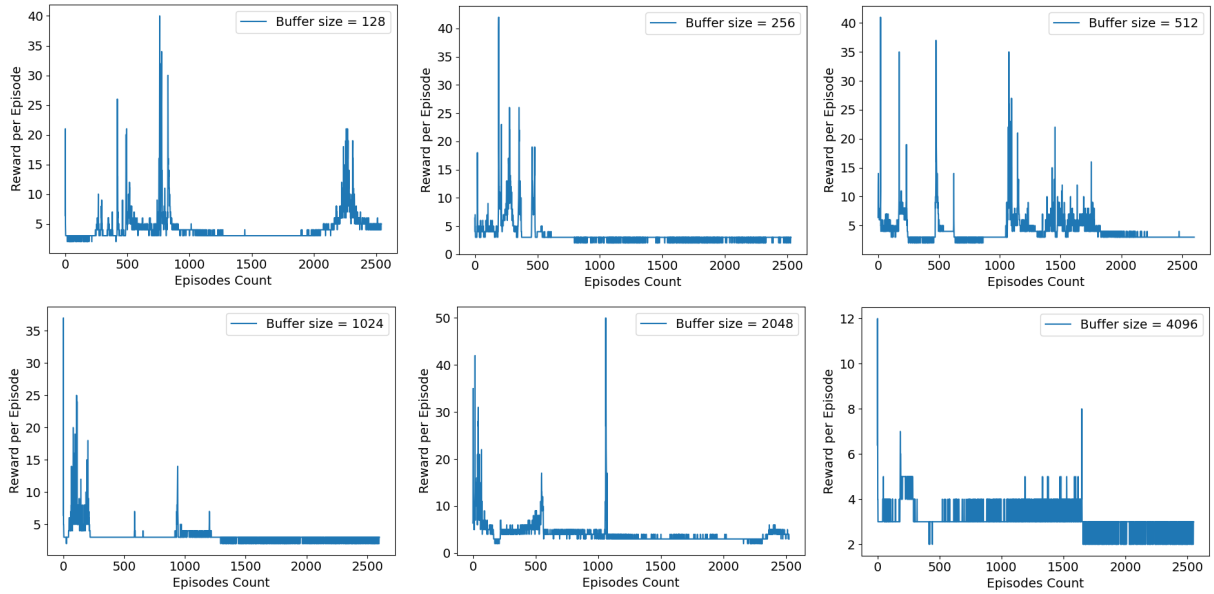

Figure 12. Generative replay training sessions with different buffer sizes for the VAE.

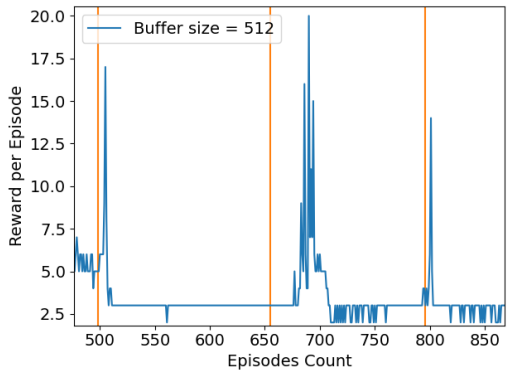

Figure 13. Detailed view into training with a buffer size of 512, VAE update moments are highlighted with orange lines.

6.5 Delayed Training

With a freeze time of 3000 steps and keeping the rest of the components same as in the other experiments, we can say that this method does show promise but by itself fails as well. Figure 14 shows the training progress of this architecture, with vertical lines indicating the freeze moments. The milestones in this experiment are as follows, $[8, 15, 20, 30, 40, 50, 60, 70, 80, 90]$. We observe that this method of training shows less of the flattening out behaviour seen in the previous experiments however, the overall performance is still not a success. There is no gradual and sustained increase of average rewards, and the milestones are reset many times as the policy degrades. This might indicate the limitations of VAEs as generative replay models in this situation since the VAE should have enough training time to catch up in this experiment.

7. DISCUSSION AND CONCLUSION

7.1 General Conclusion

This paper has explained and then presented four experiments related to generative replay in deep reinforcement learning. This is still an infant area of research so we have explored various methods in order to improve the performance of a generative replay system. The first research question stated in the introduction has been addressed as the experiments suggest that VAE might not be as feasible as assumed for generative replay. The results of the methods can be interpreted as inconclusive since visible improvement or hindering was not obvious from the data. The future paths for research is addressed below with five suggestions.

7.2 Shortcomings

One of the main weak points of this experiments is the environment used to collect data and investigate methods. The inverted pendulum is a control task that heavily relies on numerically accurate actions. Since balancing a stick is a sensitive task, even the smallest deviations in the actions the TD3 policy suggests can cause the stick to fall. This shortcoming can be addressed by applying these methods to a different set of tasks where perfect numerical accuracy is not the main issue. Another weakness of this study is the computational limitations. Many of the experiments are not run to the full extend so some information might not be visible in the figures. This is addressed mainly by running one of the graphs in the experiments to the full length and making sure that the trends are not misrepresented in the figures, however a detailed study can be made with better computing resources.

In any case, the flattening behaviour in every training scenario seems unavoidable. If the system is ran long enough, the policy of TD3 will eventually produce the worst possible actions consistently.

7.3 Value of this study

In the end, we experimented with many ways in which an artificial system such as this can be improved. The data and the knowledge gathered in this work can be quite powerful in guiding future studies that follow up. The fact that the tested methods have resulted in semi working solutions shows the potential of generative replay.

7.4 Future Work

Due to the exploratory nature of this paper, we did not present overwhelmingly positive results. However, this research has led to valuable information in the sense that many open problems have been generated. This section

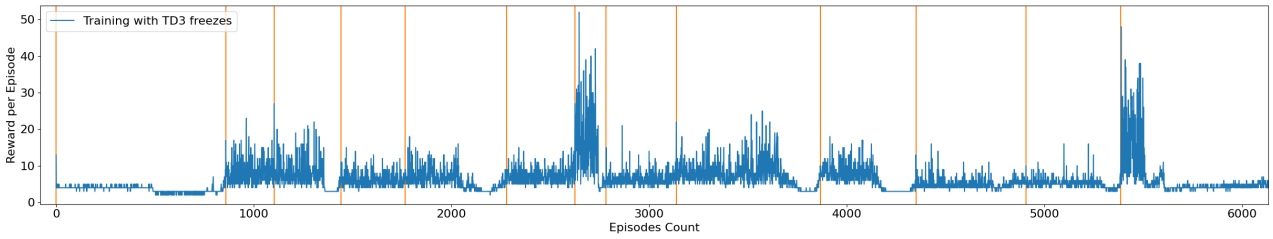

Figure 14. Delayed TD3 training. The orange lines show the moment where TD3 training was frozen.

lists and explains all the open problems that this paper has identified and what are some of the immediate experiments that can follow this work.

7.4.1 *Latent Distribution*

In all the experiments in this paper the distribution dedicated to the latent space was a normal distribution. However, there are no strict reasons why this should be the case. Logically, it makes sense to have some sort of normalization in the latent space for better sampling and better sampling should improve the training process. Further research could be conducted with different distribution models or again with normal distribution but with a deeper network to provide better resolution on the generated data.

7.4.2 *DRL Loss in Generative Replay*

One of the ideas that have not been tested in this work is to incorporate the loss or the rewards of the deep reinforcement learning agent in the training of the generative model. With all the methods in this paper they train and act independent of each other in all the explicit ways. Of course, the fact that a better DRL result gives different data to the generative memory means they interact, but they do not aim to optimize each other. If a measure of how well the DRL agent is doing is included in the training process of the generative memory, it might force the network to produce samples that help training in a more direct way. The hardship with this idea is that they train in separate intervals. As the DRL agent trains the generative memory only samples and as the generative memory trains the DRL agent is dormant. Training them at the same time without using some memory element is challenging but can be promising.

7.4.3 *Compression Instead of Generation*

The rationale behind the idea of using generative models is to decrease memory usage in order to increase scalability. Another way to possibly achieve this is to not generate new experiences but to compress the ones that would be saved in the experience replay. Autoencoders, as mentioned earlier, have the capability to reduce a given feature size to a smaller latent size and then decompress them to great resolutions. An experimental setup can be defined where the collected experience in the DRL environment are used to train an autoencoder. In addition, instead of storing the whole experiences in memory, a system can be made to store the latent representations, compressed experiences, in memory and use decompression before training to achieve less memory usage. Furthermore, experiments can be made with the autoencoder only targeting the state tuples and keeping the resolution of the other features intact. This approach might prove to be the most stable and feasible.

7.4.4 *Better Scheduled Training*

In the last experiment, a smarter schedule of training is investigated. This method might be unsuccessful to present gradual and stable improvement in the training, however it shows some promise. The main reason behind the low results of this paper is the simple way of attempting to train both the generative replay and the DRL agent at the same time. In reality, this causes them to fall into a negative feedback loop where bad experiences are sampled resulting in worse actions causing more bad experiences. Detecting when to train the generative replay and when to train the DRL agent is a topic worth investigating on its own. Ideally a dynamic method of scheduled training is needed to perform better.

7.4.5 *Different Model Than VAE*

One last future experiment can be to try other generative models such as Generative Adversarial Networks (GANs) or Restricted Boltzmann Machines (RBMs) with the knowledge from this study. Since this paper did not focus on any other model however, we cannot comment much on how promising this change would be.

8. REFERENCES

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[2] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney. Revisiting fundamentals of experience replay, 2020.

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

[4] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[5] H. Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 2613–2621. Curran Associates, Inc., 2010.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[7] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[8] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[9] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[10] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, May 1992.

[11] R. Liu and J. Zou. The effects of memory replay in reinforcement learning, 2017.

[12] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109 – 165. Academic Press, 1989.

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

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

[15] A. Raghavan, J. Hostetler, and S. Chai. Generative memory for lifelong reinforcement learning, 2019.

[16] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay, 2017.

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

[18] C. Watkins. Learning from delayed rewards. 1989.

[19] L. Weng. Policy gradient algorithms. *lilianweng.github.io/lil-log*, 2018.