Tampere University

Santeri Heiskanen

INTERMEDIATE GOAL GENERATION FOR OFF-POLICY REINFORCEMENT LEARNING METHODS

Bachelors degree
Faculty of Information Technology and Communication Sciences
Examiners: Okko Räsänen
May 2022

ABSTRACT

Santeri Heiskanen: Intermediate goal generation for off-policy reinforcement learning methods
Bachelors degree
Tampere University
Bachelor's Programme in Computing and Electrical Engineering
May 2022

Reinforcement learning is a branch of machine learning, which has seen increasing interest in the last few years. Reinforcement learning tries to teach an agent to complete a task in an optimal way. The optimality is defined through reward function that signals the desired behaviour to the agent. However, defining reward function is often difficult, which has led to numerous attempts to automatize the process so that there is no need to define such function. In this work, an existing goal generation algorithm for an on-policy reinforcement learning method is applied to an off-policy method. This work examines if the algorithm works with the off-policy method without large modifications, and if the algorithm's efficiency improves when used in conjunction with the off-policy method.

The theoretical part explains the basics of generative adversarial networks, which are a key part of the algorithm. Additionally, fundamental concepts related to reinforcement learning are presented. The used algorithm is described in detail. Generic, simple reward function is used in-place of task specific, complex reward function. To avoid the problem where the agent receives little to no rewards, the generative adversarial network is used to generate intermediate goals for the agent. The intermediate goals are used to guide the agent towards completing the original task.

To evaluate the performance of the algorithm, simple, yet illustrative task is used. In this task, a spider robot tries to reach another end of a U-shaped maze. The results from the experiment show that the algorithm requires modifications before it can be applied to off-policy methods successfully. The study concludes that more robust initialization procedure should be constructed to ensure that enough rewards are received in the beginning of the algorithm. Additionally, the exploration policy of the off-policy method is found to be insufficient for this task. The efficiency of the algorithm with off-policy methods is left as open question, as the experiment did not provide results that could answer the question directly. However, the experiment showed that the efficiency of the algorithm is poor overall, and thus more work is required to make the algorithm suitable for real-world problems.

Keywords: Reinforcement learning, curriculum learning, off-policy, automatic curriculum, generative adversarial network

TIIVISTELMÄ

Santeri Heiskanen: Välimaalien generointi menettelytavan ulkopuolista tutkimista hyödyntäville vahvistusoppimisen metodeille
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan kandidaattiohjelma
Toukokuu 2022

Vahvistusoppiminen on koneoppimistekniikka, johon viime vuosina kohdistunut lisääntynyttä kiinnostusta. Vahvistusoppimisen tarkoituksena on opettaa agentti suorittamaan annettu tehtävä optimaalisella tavalla. Optimaalisuus määritellään palkkiofunktiolla, jonka avulla agentille kerrotaan halutta käyttäytymistapa. Palkkiofunktion määrittäminen on kuitenkin usein haastavaa. Tämä on luonut tarpeen metodille, jolla agentti voidaan opettaa suorittamaan tehtävä ilman tehtäväkohtaista palkkiofunktiota. Tämä tutkimus soveltaa jo olemassa olevaa välimaaleja generoivaa algoritmia vahvistusoppimismetodiin, jossa ympäristön tutkiminen on toteutettu erillään tehtävän suorittamista varten opittavasta käytöksestä. Tutkimus pyrkii selvittämään, toimiiko algoritmi kyseisen vahvistusoppimismetodin kanssa ilman suuria muutoksia, ja paraneeko algoritmin tehokkuus, kun sitä käytetään kyseisen metodin kanssa.

Tutkimus selittää tärkeimmät konseptit generatiivisista kilpailevista verkoista, jotka ovat keskeinen komponentti algoritmissa. Tämän lisäksi vahvistusoppimisen tärkeimmät käsitteet on esitelty, ja käytetty algoritmi on kuvailtu yksityiskohtaisesti. Algoritmi vaihtaa tehtäväkohtaisen palkkiofunktion yksinkertaiseen ja geneeriseen palkkiofunktioon. Välttääkseen ongelman, jossa agentti ei vastaanota riittävästi palkkioita tehtävän oppimiseksi, algoritmi generoi välimaaleja generatiivisen kilpailevan verkon avulla. Generoidut välimaalit ohjaavat agenttia kohti sen lopullista tavoitetta.

Tutkimus arvioi algoritmin tehokkuutta esitetyn vahvistusoppimismetodin kanssa käyttämällä yksinkertaista, mutta havainnollista tehtävää, jossa hämähäkkirobotin tulee saavuttaa U:n muotoisen labyrintin toinen pääty. Tulokset osoittavat, että algoritmi ei pysty opettamaan agenttia onnistuneesti esitetyn vahvistusoppimismetodin kanssa. Tutkimus toteaa tulosten perusteella, että algoritmi tarvitsee vankemman käyttöönottomenettelyn varmistuakseen siitä, että agentti vastaanottaa tarpeeksi palkkioita opetusprosessin alkuvaiheissa. Lisäksi käytetyn vahvistusoppimismetodin ympäristön tutkimusmenetelmä todetaan riittämättömäksi tutkimuksessa käytetyssä tehtävässä. Algoritmin tehokkuus esitellyn vahvistusoppimismetodin kanssa jää avoimeksi kysymykseksi, mutta tutkimus toteaa, että algoritmin yleinen rakenne vaatii muutoksia, jotta tehokkuus voisi saavuttaa riittävän tason monimutkaisissa tehtävissä.

Avainsanat: Vahvistusoppiminen, menettelytavan ulkopuolinen oppiminen, automaattinen välimaalien luonti, generatiivinen kilpaileva verkko

CONTENTS

GLOSSARY

DDPG	Deep deterministic policy gradient
DPG	Deep policy gradient
DQN	Deep Q network
GA	Genetic algorithm
GAN	Generative adversarial network
GOID	Goals of intermediate difficulty
LSGAN	Least squares generative adversarial network
MDP	Markov decision process
RL	Reinforcement learning
TRPO	Trust region policy optimization

1. INTRODUCTION

Reinforcement learning is branch of a machine learning, which has seen increasing interest in the last few years. The reinforcement learning techniques have been applied to many different tasks, such as robot control[20],[25], games[22], [33] and estimating stock markets[27] to name a few. Despite its success in diverse range of applications, the reinforcement learning methods have their own share of problems. One fundamental problem with reinforcement learning is the difficulty of designing a suitable reward function that describes the desired behaviour. This is a task specific problem, meaning that the problem must be solved every time the method is applied to a new application. Recently, promising work has emerged on creating automatic methods that offer a way to use reinforcement learning techniques without designing a unique reward function.

In this study, an existing algorithm is taken, and it is applied to an off-policy method. The algorithm, proposed by Florensa et al. [9], consists of a generative adversarial network, which is used in conjunction with a reinforcement learning method to avoid designing a reward function. However, the algorithm was originally used with an on-policy method. In this study, the possibility of applying this algorithm to an off-policy method is explored. In particular, this study tries to answer the following questions:

- Does the algorithm work with an off-policy method without excessive rework? And if not, what kind of modifications would be required?
- Does the efficiency of the algorithm improve when used in conjunction with an off-policy method?

The latter question is motivated by the fact that off-policy methods are known to be often more sample efficient than on-policy methods. That is to say, the off-policy methods require often less steps than on-policy methods to learn to complete a given task.

The structure of this thesis is the following: the Chapter 2 introduces the relevant theory behind the generative adversarial networks and reinforcement learning. Chapter 3 presents some previous work that try to avoid the requirement to design a specific reward function. Chapter 4 goes over the used algorithm in details, and the presents the experiment setup used to test the method capabilities. Lastly, Chapter 5 presents the results of the experiment and provides some discussion on the future research directions around the topic.

2. THEORY

In this chapter, we will look at the underlying theory behind the methods used in the Chapter 4. The objective of this chapter is to give the reader deep understanding of the mathematical foundations of the used methods, and their possible strengths and weaknesses. This includes a deep dive into generative adversarial networks and basics of reinforcement learning.

2.1 Generative adversarial networks

Generative adversarial network (GAN) is a generative neural network architecture introduced by Goodfellow et al.[11] in 2014. Originally, the GAN was used to generate images that were similar to the images in the training set. After the original GAN formulation, many variants have been introduced. This has allowed the GAN to be applied to multitude of problems, such as resolution up-scaling, text to image synthesis and text generation. The Section 2.1.1 introduces the basic concepts of the GAN, Section 2.1.2 describes the most relevant problems and benefits of the GAN. Lastly, Section 2.1.3 presents the original GAN in more details and describes a variant that introduce changes to the GAN's loss function.

2.1.1 Learning generator function using adversarial networks

The idea behind GANs is to learn a *generator function (G)* that maps data from distribution $\mathcal{Z}$ to a distribution $G(\mathcal{Z})$, which should be similar to the original data distribution $\mathcal{X}$. This problem formulation has an implicit assumption that there exists at least one such vector $\mathbf{z} \sim \mathcal{Z}$ for each sample $\mathbf{x} \sim \mathcal{X}$. Usually, the $\mathbf{z}$ is called latent vector, and $\mathcal{Z}$ is called latent space. Generally speaking, the dimensionality of the latent vector is usually smaller than the dimensionality of the sample vector $\mathbf{x}$, as most of the data don't actually contain as high dimensional information than they seem to. Intuitively, latent-space can be seen as a distribution, that can be used as a input to generate all the necessary information of a more complex distribution $\mathcal{X}$. Often in practice, $\mathcal{Z}$ is a univariate Gaussian distribution, but any tractable distribution could be used. Now, if the generator exists, one can create new samples of $\mathcal{X}$ by sampling $\mathbf{z} \sim \mathcal{Z}$, and computing $G(\mathbf{z})$.[29, pp. 2-3] In addition to the generator, the GAN contains also another network, *Discriminator (D)*. The responsibility of the discriminator is to recognize if a given sample came from the original data distribution, or if it was generated. In the original paper, the discriminator outputted the probability of a given sample being from the original data, rather than being generated by the generator network[11].

Originally, the idea of training a GAN was described as a two-player game, where the 2 networks are put against each other. This leads to situation where the generator tries to become better at producing samples that will fool the discriminator, while simultaneously the discriminator tries to become better at distinguishing which samples are real and which not.[11] In the optimal state of this game, the generator cannot get any better at producing samples without making the discriminator worse at recognizing the generated samples from the real samples, and vice-versa. The optimal state is called *Nash*-equilibrium, and reaching this state is often difficult, as the problem is a saddle-point problem, rather than a minimization problem. Unfortunately, saddle-points are often unstable, and difficult to approximate numerically[29, pp. 18].

There is also another way of looking at the process of training a GAN. Intuitively, it is clear that if one wants to find a generator function that maps data from the latent space $\mathcal{Z}$ to a distribution that is similar than the original distribution $\mathcal{X}$, one must be approximate how "close" the generated distribution $G(\mathcal{Z})$ is to the original distribution $\mathcal{X}$. The "closeness" is often measured with a distance function, and thus the smaller the distance between these 2 distributions is, the better the generated samples must be. For this reason, the training objective of GAN can also be seen as minimizing the distance between the 2 distributions. In practice, this means that the used loss function should effectively measure the distance between the distributions, and as it turns out, choosing this distance function is one of the most difficult tasks of training a GAN.[29, pp. 17] These are called statistical divergences, and several variants of the GAN have been proposed with different distance metrics[30, pp. 29], and one such variant will be introduced in Section 2.1.3.

2.1.2 Drawbacks and benefits

Successfully training a GAN is notoriously difficult, as the training process of the networks is often unstable and can lead to mode collapse. The mode collapse is a phenomenon, where the generator starts to generate a very small set of samples, which won't represent the whole distribution of the original data set. In the extreme case, the generator might produce the same single sample every time. Intuitively this can be seen as the result from situation, where the GAN always tries to produce samples that are the most plausible for the discriminator. The networks are also known to suffer from vanishing gradients. [30, pp. 8 - 9]

Besides the difficulties mentioned above, the GAN has other problems as well. Firstly, it requires a large amount of data to generate samples that have good coverage of the original data distribution. Even if the generator doesn't suffer from extreme mode collapse during the training, it might still produce samples with limited diversity. Additionally, the GAN suffers from same difficulties than other generative models. One large problem with generative models is the evaluation of the results, which is often qualitative. For example, visual inspection is commonly used when working with images. In addition to being time-consuming, it is also subjective, which makes it difficult to compare results from different methods. It also cannot describe the characteristics of the produced distribution, which is important for unsupervised machine learning models. [30, pp. 9-10]

Despite its limitations, the generative adversarial networks have some desirable properties. Firstly, the structure of the network is quite flexible, and it can accommodate almost any differentiable function. Secondly, GANs don't require the use of Markov chains, which allows them to present more complex distributions. [11, pp. 7] Additionally, the GANs are so called likelihood-free models, which means that they don't make any assumptions of the underlying model [29, pp. 17]. On top of this, GANs can model high-dimensionality distributions, which makes it possible to apply GANs to new application-fields, such as video and audio, which have been historically difficult to model[30, pp. 5].

2.1.3 Alternative distance metrics

As mentioned in the Section 2.1.2, the GANs have had their fair share of problems, many of which are related to the training process. Some mathematical background behind these problems will be presented below, and a variant will be introduced, which tries to improve the original approach by introducing an alternative distance metric.

The Equation (2.1) describes the originally proposed cross-entropy loss function for the GAN.

$$\underset{G}{min}\ \underset{D}{max}\ V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[log\ D(x)] + \mathbb{E}_{x\sim p_z(z)}[log(1-D(G(z))], \tag{2.1}$$

where G is the generator network, D is the discriminator network. As mentioned before, this problem is a saddle point problem, and reaching its optimal state is often difficult in practice. It can be shown that solving this problem is similar to minimizing Jensen-Shannon divergence, which can be used to measure the similarity of 2 distributions.[11] Unfortunately, this formulation makes the training process of GAN unstable, as Arjovsky and Bottou[2] note. The gradients of the generator can start to vanish, if the discriminator learns to recognize generated samples from original ones faster than the generator can learn to produce plausible looking samples. For this reason, one must very carefully balance the training between the generator and discriminator, which in turn makes GAN's training process extremely difficult.[2, pp. 6-7]

In 2017, Mao et al.[21] released a variant of the original GAN, called least squares GAN

(LSGAN). The method adopted a least squared based loss-function formulation, which is presented in Equation (2.2).

$$\begin{aligned} \underset{D}{min}\, V_{LSGAN}(D) &= \frac{1}{2}\mathbb{E}_{x\sim p_{data(x)}}[(D(x)-b)^2] + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z))-a)^2] \\ \underset{G}{min}\, V_{LSGAN}(G) &= \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z))-c)^2], \end{aligned} \tag{2.2}$$

where a and b are the used labels for generated and real data, respectively. The value c presents the target label/value that the generator wants the discriminator to output for the generated data. It can be shown that minimizing the Equation (2.2) is related to minimizing the Pearson χ^2 divergence. This, however, adds the following constraints to the values of a, b and c: $b - c = 1$ and $b - a = 2$. Original paper fulfilled this requirement by setting values of a, b and c to -1, 1 and 0 respectively. The LSGAN brings 2 main improvements compared to the original GAN. Firstly, Equation (2.2) allows to penalize samples that are mislabeled by the discriminator (which is the generator's target), but which are still very dissimilar to the samples from the original distribution. Additionally, penalizing the samples that are slightly above the threshold that makes the discriminator to mislabel them, can generate more gradients while updating the generator. This alleviates the problem of vanishing gradients.[21, pp. 6-7]. The algorithm used in this work adopts a modified version of the LSGAN, which will be described in Section 4.2.

2.2 Reinforcement learning

The machine learning methods are often classified into supervised and unsupervised machine learning methods. However, there is also a third branch of machine learning methods, called *Reinforcement learning (RL)*, which uses an interactive learning process.

2.2.1 Learning through trial and error

Reinforcement learning methods are a class of machine learning algorithms, where the basic idea is to learn an optimal way of completing a task, through trial and error. This is achieved by letting an *agent* to interact with its surrounding *environment*. To interact with the environment, the agent makes *actions*. The agent should receive feedback on its actions, that describe how desirable the taken action was. The feedback is often referred to as *reward*, and it adds indirection to the problem formulation, as the solution to the problem is not given explicitly to the agent, but rather it is described through the feedback that the agent receives.[17, pp. 1]

The goal of reinforcement learning is to identify an optimal policy π^*, which maximizes the cumulative reward for the agent. The policy is a mapping from states to actions, that tries to make the optimal action on each state, while taking into account the whole lifetime of the agent[17, pp. 3]. Intuitively, the policy can be seen as ruling that the agent follows when making decisions. The policy can be either stochastic or deterministic. Exploration of the environment is vital part of learning the policy, as without exploration the agent cannot learn the relations between different states, actions and rewards. This demand for exploration leads to a trade-off, where the agent needs to decide, if it takes risks by exploring more unknown states and actions, or if it sticks only to actions that it knows well. This is commonly referred as exploration-exploitation trade-off. How this trade-off is approached leads to 2 distinct types of methods, *on-policy*, and *off-policy*. In on-policy methods, the actions are picked using the current policy. To achieve the better coverage of the environment, the exploration must be built into the policy, which in turn determines how fast the policy can improve. In contrast, the off-policy methods use an independent strategy for exploration, which differs from the policy that is the target of the learning.[17, pp. 5]

As mentioned above, the optimal policy should maximize the expected cumulative reward over the lifetime. However, there are many ways in which the expected reward can be defined. Many times, the agent has an *episodic* lifetime, where one episode contains certain amount time steps. The episode ends when the agent succeeds in the task, or when the maximum amount of time steps has been taken. In episodic lifetime, after an episode ends, the environment is reseted to its original state, and the task is restarted. In this case the optimal policy maximizes the expected reward over one episode[17, pp. 1241]. More generally, this can be seen as an *finite-horizon model*, where the agent tries to maximize the expected reward over the next h steps. The Equation (2.3) shows the formula to be maximized[16, pp. 240].

$$J = \mathbb{E}(\sum_{t=0}^{h} r_t) \tag{2.3}$$

However, in some scenarios, the start and end points of the task cannot be clearly defined, as the task might be a continuous process. In these cases, a weighted sum reward can be applicable [17, pp. 1241]. More generally, this scenario can be seen as a *infinite-horizon model*, where each reward is multiplied by a *discount factor* γ, $0 \leq \gamma < 1$. The Equation (2.4) shows the maximized equation[16, pp. 240].

$$J = \mathbb{E}(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{2.4}$$

This allows the agent to put less emphasis on the future rewards that are far-away and focus more on the short-term rewards. The value of the discount factor γ has a significant effect on the learned policy, as policies with small discount factors become greedy, which tends to undermine the long-term performance. Additionally, too small discount factor can also introduce instability to the optimal policy. [17, pp. 1242]

It is intuitively clear, that while the agent tries to explore the environment, and learn the optimal policy, it will make a sequence of decisions, that determine which action is chosen at any given state. This can be modelled more formally as Markov Decision Process (MDP)[23, pp. 3]. In this MPD, the agent is in a given *state* s, where it will take an action a, which might change its state to a new state s'. Lastly, agent receives a reward r as feedback for the action. This process is iterated over until the stopping criteria is reached. This could be the completion of the task or reaching the maximum number of allowed steps. The MPD can be presented as a tuple $\langle S, A, \Phi, R\rangle$ [23, pp. 775], where

- S is finite set of states $(s_i \in S, i = \{1, \cdots n\})$, where the agent resides
- A is a state dependent, finite set of actions $(a_j(s_i) \in A, j = \{1, \cdots, m\})$
- $R : S \times A \rightarrow \mathcal{R}$ is reward function, which maps state-action pairs to the *reward*-space (often $\mathbb{R}$), which describes how desirable a given pair is.
- $\Phi : S \times A \rightarrow S$ is a state transition function, which describes the probability of reaching a new state $s' \in S$ when action a is taken from state s.

The MDP formulation makes an assumption that the new state s' and the received reward r depend only on the previous state s and action a. Most of the RL methods rely on this assumption, even if it doesn't get fulfilled in real world cases. [36, see pp.1242, 17]

One common problem in reinforcement learning is the question on how to choose the appropriate reward function $R(s, a)$. Choosing a descriptive reward function is important, as the reward function is the only way to signal to the agent what the desired behaviour is. [17, pp. 1252] The first though might be to just give the agent a reward only when it has completed the task, and otherwise give it a zero reward or a small penalty. However, to effectively learn the desired policy, the agent must be able to observe some variance in the received reward. As it turns out, when using *sparse reward functions*, the agent might receive a meaningful change in the reward so rarely, that it cannot effectively learn the desired policy. For this reason, the reward function should provide the agent with intermediate goals, that provide the agent with information how close its behaviour is to the desired behaviour. This process of defining reward function that provides the agent with reasonable rewards throughout of the learning process is called *reward function shaping*. However, designing such function is difficult in real-life. There is no general formula for creating adequate reward function, and thus it requires lot of application-specific knowledge. [17, pp. 1253].

2.2.2 Approaches

Finding the optimal policy π^*, can be seen as an optimization problem, which leads to 2 possible formulations that can be used to reach the optimal policy. Depending on the used formulation, the methods are classified to *Value-based* and *Policy-based* methods. In the value-based methods, the models try to find a solution, or an approximation of the solution to the Equation (2.5), which is also known as value function.

$$V^*(s) = \underset{a^*}{max}[R(s,a^*) - \hat{R} + \sum_{s'} V^*(s)\Phi(s'|a^*,s)], \tag{2.5}$$

where the $\hat{R}$ is the average reward when starting from state s and $\Phi(s'|a^*,s)$ is the probability of reaching state s' when taking action a from state s .[17, pp. 1243-1244] The meaning of the Equation (2.5) can be interpreted as follows: when starting from an arbitrary state, choosing only optimal actions will result in optimal solution, or in other words, the policy is optimal if and only if each action is optimal. Thus, by first solving the Equation (2.5), the value-based methods can construct the optimal policy. On the other hand, the policy-based methods try to solve the optimal policy directly, which has been historically more difficult problem, as the policy-based methods often converge to local optimum rather than the global one[23, pp. 778]. The reason behind this is that many policy-based methods only consider the neighbourhood of the current policy. In addition to value-based and policy-based methods, there are some methods that combine these 2 approaches. These are called *Actor-critic* methods, where the value function acts as the critic, which observes and estimates the performance, while the actor tries to find the optimal policy. Based on the current policy's performance, the critic can decide if the current policy needs to be updated. [17, pp. 1248]

In addition to being classified by the approach used to solve the problem, the reinforcement learning methods can also be classified based on the methods requirement for transition function $\Phi(s'|a,s)$, and the reward function $R(s,a)$. The *model-based* methods require the transition function and reward function. This allows them to avoid interactions with the environment and learn the optimal policy directly using the model. [14, pp. 5]. The model doesn't have to be known beforehand, as it can be learned incrementally from the data. However, one should note that building such model requires lot of data. [17, pp. 1244] The *model-free* methods don't require the knowledge of the transition function, and instead rely on the interaction with the environment to improve the policy incrementally[14, pp.5].

3. RELATED WORK

In this chapter, other approaches for avoiding difficult reward function shaping are presented. As mentioned before, choosing an effective reward function requires extensive knowledge of the environment, and even then, choosing such function might not be straightforward in all cases. This has led to development towards methods that don't require carefully constructed reward function to learn to complete the presented task.

One popular approach is called *Curriculum learning*, which refers to a training process, where the agent is presented with set of increasingly difficult intermediary goals that guide the agent to learn to complete the actual task. The usage of intermediate goals allows the agent to receive enough reward signals to learn to complete the tasks, even in sparse, long horizon tasks. The idea of curriculum learning was first introduced to reinforcement learning by Bengio et al. [5] in 2009. In their original paper, the curriculum was hand-designed. However, after this, there has been multiple attempts to create systems that can generate suitable curricula automatically.

In 2019, a technique called *Curriculum goal masking* was introduced by Eppe et al.[8]. This work approached the problem of generating suitable curricula by applying a mask to the original goal to generate sub goals for the agent. In more details, a binary vector was applied to the vector presenting the goal, and the sub goal consisted only the elements that were masked. The experiments done by the authors showed increased performance compared to the baseline methods. However, the method was tested only in 3-dimensional goal-space, and it is unclear how well the method scales to higher dimensional goal-space, where the amount of possible sub goal combinations grows. Interestingly, Florensa et al.[10] noticed in 2018 that the curricula can be generated in reverse. That is to say, the authors started training the agent from positions close to the final goal. The policy is then improved by gradually moving the agent's starting position away from the final goal, making the task increasingly more difficult for the agent. The authors argue that training the agent from multiple positions improve the robustness of the final policy. The experiments show that the method's effectiveness decreases when the task gets more complex. Additionally, the method is only applicable to on-policy methods.

He et al.[12] took a similar approach to this work and created a goal-generator to produce intermediate goals for the agent. Their work used another MDP to generate the goals, as opposed to the GAN used in this work. The authors presented an algorithm that trained both policies simultaneously. The experiment shows an improvement compared to the baselines, but the authors note that the performance of the method seems to be dependent on the hyperparameter which controls the size of the horizon used by the generator. The optimal value for this hyperparameter seems to be environment dependent, which complicates the process of applying the method to new tasks. More recently, Song and Schneider [34] used a genetic algorithm (GA) to create scenarios that can be used to learn the desired policy. The genetic algorithm is a non-parametric optimizer, which decreases the reliance on application specific knowledge. The algorithm starts by choosing scenarios where the agent fails often, and uses those as the starting point for the GA. The scenarios generated by the GA are then evaluated using the current policy, and if the agent cannot solve the scenario, it will be trained on that scenario to improve its performance. The experiments show that method improves the robustness of the learned policy, but the authors note that the method is computationally expensive compared to other methods.

Another common approach is called *intrinsic motivation*, which tries to improve the agent's exploration in the environment. The idea behind the intrinsic motivation is that an agent should make an action for its own sake, out of curiosity, rather than for achieving certain task. The intrinsically motivated agent should be more efficient at exploring, as this often leads to better performing policy, even if the reward function doesn't necessarily motivate the agent to explore more. The concept of intrinsic motivation stems from the behaviour of animals, and it was first introduced to the reinforcement learning context by Barto et al.[4].

In 2013, Baraners et al. [3] presented a method called SAGG-RIAC, which motivated the agent's exploration based on its competence in certain regions. The method was able to achieve better exploration than the method it was compared to, but it requires that one defines a function that evaluates the agent's competence in a given region. This raises complexity of using the method. Interestingly, Sukhbaatar et al.[35] combined the idea of intrinsic motivation and curriculum learning in method called asymmetric self-play. In their work, the authors used 2 "minds", Alice and Bob to guide the same agent. Alice's responsibility was to find the simplest task that Bob cannot complete. After the Alice has found the task, Bob's job is to start from the position where Alice stopped and solve the task in reverse order to reset the environment to its original state. The novelty of the method comes from the fact, that the method doesn't require external guiding signal from the environment, and instead an internal reward is used. The authors note that if the task is highly asymmetric, i.e. it is significantly easier to complete in the original order than in reverse or vice versa, then the method might suffer from poor performance. To fix such issue, the authors propose that in such cases the environment is reseted and Bob is set to complete the same task than Alice. The biggest weakness of the method is that is relies on the assumption that the task is either reversible, or that the environment is resettable.

More recently, Bougie and Ichise[6] presented goal-spaced curiosity module (GoCu), which added curiosity based intrinsic reward to the external reward received from the environment. The intrinsic reward was used to guide the agent to explore states that had high probability for learning new skills. To address the problem of sparse rewards, the states visited during last episode were sampled to act as intermediate goals. The method was used only to train on-policy algorithms and it is uncertain how the method would adapt to off-policy methods. Lastly, Li et al. [18] introduced an anchor-based system, where so called anchor points were chosen from states that the agent had already visited. Then, an intrinsic reward is added, which motivates the agent to move away from the states it has already reached. This allows the agent to explore new states efficiently. However, the authors used environment specific functions to create the rewards, which lessens the generality of the method.

4. METHODOLOGY

This chapter will introduce *Goal-GAN* algorithm, which mimics very closely the work of Florensa et al.[9]. The algorithm is extended and tested with off-policy RL agent, which is something that was not explored in the original paper. The Sections 4.1 and 4.2 will be citing their work, unless explicitly stated otherwise. The Goal-GAN algorithm tries to improve the usability of reinforcement learning by addressing one of the fundamental problems: The difficulty of designing a suitable reward function.

4.1 Objective

Recall from Section 2.2, that traditionally the goal of RL methods is to find the optimal policy π^*, such that the return value from reward function R is maximized over the whole lifetime of an agent. In the following sections, a slightly different objective is adopted. Instead of maximizing return of a single reward function, the optimal policy π^* should be able to consider a range of goals, where each goal g has its own reward function r_g. Here, a simple indicator function presented in Equation (4.1) is used as the reward function r_g for each goal.

$$r_g(s, a, s') = \mathbb{1}\{s' \in S_g\} \tag{4.1}$$

In this study S_g is defined by $S_g := \{s : d(f(s),\ g) \leq \epsilon\}$. The $d(\cdot, \cdot)$ is a distance function in the goal space, $f(\cdot)$ is a function mapping states to goal space and ϵ is the allowed tolerance in the distance. The agent is trained in an episodic setting, where each episode is terminated if $s \in S_g$, or if the maximum amount of time steps T has been used. The reward over the whole episode is presented in Equation (4.2)

$$R_g = \sum_{t=0}^{T} r_{g,t}, \tag{4.2}$$

which is a binary random variable. When actions are sampled using a policy π, the expected return for that policy $R(\pi)_g$ can be interpreted as the probability of reaching a goal g at most T time steps.

Now, the objective is to find policy π^*, that achieves high reward for as many goals as possible. In other words, the policy must consider a set of goals, and try to perform optimally in regard to each goal. Suppose that there exists a distribution of goals $p_g(g)$, which the agent should be able to reach with good performance. Then the objective becomes to find a policy π^*, that satisfies the Equation (4.3)

$$\pi^* = \underset{\pi}{arg\ max}\ \mathbb{E}_{g \sim p_g} R_g(\pi) \tag{4.3}$$

If one recalls from the above that the expected return with a given policy $R_g(\pi)$ can be seen as a probability of reaching goal g, then the Equation (4.3) can be interpreted as the *average* probability of reaching g over all goals in p_g. That is to say, we are looking for a policy π^*, that maximizes the probability of reaching arbitrary goal $g \sim p_g$.

The following assumptions are made[9]:

1. If sufficient amount of goals from some area of the goal-space are used to train a policy, then that policy can learn to interpolate other goals within that area.
2. If policy is trained on set of goals, then it will provide a good starting point for learning other close-by goals.
3. For each reachable goal, there exists a policy which can reach the goal consistently.

The first assumption means that one can expect the policy to generalize to goals that are similar to those it has been trained with. The second assumption assures that the trained policy should be able to reach close-by goals sometimes but might not be able do so consistently. This is an intuitive assumption to make, but one that is crucial for the algorithm, as it relies on improving the policy iteratively. The last assumption is often implicitly assumed, and it ensures that the policy described in Equation 4.3 exists.

As mentioned in the Section 2.2, one fundamental challenge in RL is designing a suitable reward function. The Goal-GAN algorithm avoids this problem by using only the indicator function (4.1) to learn the optimal policy. The indicator function just requires that one defines when the task is completed, which is often easier than describing if certain action moved the agent closer to the solution.

4.2 Intermediate goal generation to optimize the policy

The Goal-GAN algorithm tries to train a policy using a general indicator function. A common problem with sparse reward functions is that the agent doesn't receive enough reward signals to learn the desired policy. To overcome this problem, the algorithm uses a GAN to generate intermediate goals for the agent. If the generated goals have correct difficulty level, then the agent should be able to reach them using the current policy. This should allow the agent to receive enough rewards to improve the policy. The exact steps of method are presented in algorithm 1.

Algorithm 1: Generative goal learning [9]

```
Initialize the GAN
old goals $\leftarrow \emptyset$
for $i \leftarrow 1$ to $N$ do
    $z \leftarrow$ sample noise()
    goals $\leftarrow$ G(z) $\cup$ sample(old goals)
    $\pi_i \leftarrow$ update policy(goals, $\pi_{i-1}$)
    returns $\leftarrow$ evaluate policy(goals, $\pi_i$)
    labels $\leftarrow$ label goals(returns)
    (G,D) $\leftarrow$ train GAN(goals,labels, G, D)
    old goals $\leftarrow$ update replay(goals)
end
```

Before the main loop of the algorithm 1 begins, the GAN is initialized. Intuitively, it is clear that the initial policy would have hard time reaching goals that are sampled uniformly from the goal-space G. For this reason, the initialization procedure is used, as it should speed-up the learning significantly. The objective is to make a good guess of the goals that the initial policy might be able to reach. To accomplish this, goals are sampled uniformly from goal space, and the agent is set to reach those goals. As the agent explores the environment, each state that it visits is saved. These states should give a good idea on what kind of goals the initial policy is capable of reaching. To produce the initial goals, the GAN is trained using this set of states, and based on the second assumption made in the Section 4.1, the initial policy should be able to reach the goals that the GAN produces.

Suppose that one has a set of goals that the agent must reach. To proceed in the algorithm, the set of goals is used to update the current policy. During the updating process, the number of times the agent reaches a given goal is recorded. These values are then divided by the maximum number of times the agent could have visited the goal, and the resulting floating-point values are compressed to be values between 0 and 1. The resulting values are called *returns*, and they are used to classify goals based on the policy's performance.

As mentioned before, sampling goals uniformly from the goal space is hardly an effective technique due to the fact that the reward function (4.1) is sparse. To find the optimal policy, where optimality is defined as in Equation (4.3), one must modify the distribution where the goals are sampled from. This is achieved by labeling the current set of goals based on how difficult they are to the current policy, and then training the GAN using this information as guidance. The objective is to produce a set of goals that have a feasible difficulty level for the *current* policy, i.e. the policy should be able to reach the goals consistently, but not too often. This set of goals is called *Goals of Intermediate Difficulty*

(GOID), and it is defined in Equation (4.4).

$$GOID_i := \{g : R_{min} \leq R_g \leq R_{max}\} \subseteq G, \tag{4.4}$$

where R_g is the return value of goal g, and G represents the whole goal-space. Note that $GOID$ is depended on the current iteration. That is to say, it must be updated during each iteration to match the current capabilities of the policy. The values R_{min} and R_{max} are used to control the difficulty of the goals that are considered to be in $GOID$. The justification for these parameters is the following: Recall from Section 2.2 that a common problem with sparse reward functions is that the agent might explore endlessly in the environment without receiving any meaningful reward. As the used reward function (4.1) is a sparse reward function, this problem must be addressed, which is done by adding the minimum return value R_{min} to the $GOID$ definition. This minimum value ensures that the policy is able to reach $g \in GOID$ consistently. The upper limit R_{max} is used to discourage the GAN from producing goals that the policy can already reach easily, and rather produce goals that the policy has still difficulties to reach consistently. The values 0.1 and 0.9 were used for R_{min} and R_{max} respectively, just as in the original paper.

GAN is proposed for the goal generation. In the Section 2.1 it was noted that GAN's have become quite popular choice when a generative model is required, as they have the ability to model even complex high-dimensional distributions. This will allow the algorithm to scale to tasks containing high dimensional observation and state spaces. As mentioned in the Section 2.1, the original GAN formulation has suffered from instability during training, and thus the more stable LSGAN variant[21] is used. To accommodate information about $GOID$, the cost function of LSGAN is slightly modified, and it is shown in Equation (4.5)

$$\begin{aligned}&\underset{D}{min}\, V(D) = \mathbb{E}_{g\sim p_{data}(g)}[y_g(D(g)-b)^2 + (1-y_g)(D(g)-a)^2] + \mathbb{E}_{z\sim p_z(z)}[D(G(z))-a)^2]\\&\underset{G}{min}\, V(G) = \mathbb{E}_{z\sim p_z(z)}[D(G(z))-c)^2],\end{aligned} \tag{4.5}$$

where D and G are the discriminator and generator networks respectively. Compared to the LSGAN's original cost function (2.2), the new cost function has a new term, y_g. This is a binary label, which describes if a goal is in $GOID$ or not. If the goal is in $GOID$, then $y_g = 1$, and the Equation (4.5) reduces to the original cost function (2.2). The benefit of adding the binary label is that now the GAN can also learn from the goals that aren't in the $GOID$. That is to say, the GAN can utilize the information about goals that are either too easy or too difficult for the current policy. The parameters a, b and c are set to -1, 1 and 0 respectively, just like in the original paper describing the LSGAN.

Lastly, the algorithm uses a replay buffer to avoid *catastrophic forgetting* during the training process. Catastrophic forgetting (sometimes called catastrophic interference) refers to a phenomenon, where a model forgets what it has learned previously when presented with new information to learn from. That is to say, the model's performance degrades on task that were presented earlier as the learning process progresses. This problem is especially common in sequential learning processes, such as reinforcement learning. One common way to alleviate the problem is to use a replay buffer. The usage of replay buffer was firstly formulated by R. Ratcliff[28] in 1990. The replay buffer is used to store old tasks, that are then used to rehearse the model during the training process to ensure that the model doesn't forget how to complete the old tasks. To avoid a situation where the goals in the buffer are concentrated on a small area, an additional restriction was added to the replay buffer. For a goal to be added to the buffer, it must be at least a minimum distance ϵ away from the other goals in the buffer. Same distance metric $d(\cdot, \cdot)$ and minimum distance ϵ from Equation (4.2) were used.

This concludes the Goal-GAN algorithm. The algorithm uses a sparse reward function (4.1) to avoid the need to design a task specific reward function. By evaluating the policy's performance during the training and modifying the distribution where the intermediate goals are sampled, the algorithm ensures that the agent receives adequate rewards during the training. The ability to use a simple indicator function as the reward function makes the algorithm easily adaptable to large range of tasks. The choice of GAN as the generative model allows the algorithm to perform well even in complex tasks, where the observation and action spaces might be complex and high dimensional.

4.3 Agent selection

In their original work, Florensa et al.[9] mentioned that the Goal-GAN algorithm should be able to train any RL agent. The ability to train any agent, regardless of its underlying implementation is a very appealing trait, one that would make the algorithm extremely versatile. However, the authors didn't verify this claim in their original work, and instead all experiments were done using an agent based on Trust Region Policy Optimization (TRPO)[31] method. To assess the algorithm's performance using an off-policy method, this work uses an agent that is based on Deep Deterministic Policy Gradient (DDPG)[19] method.

The DDPG was introduced in 2015 by Lillicrap et al.[19], and it extends the Deep Policy Gradient (DPG)[32] framework. Additionally, the DDPG takes some motivation from the Deep Q network (DQN)[22], which operated on discrete domain, and applies its principles to continuous domain. This is achieved by adopting the *actor-critic* structure from the DPG framework. In DDPG, the actor's μ role is learn the desired policy $\pi(s|\theta^{\mu})$, which maximizes the expected reward over the lifetime of the agent. The critic's Q responsibility is to approximate the *action-value* function, which describes the expected reward over the lifetime of the agent. The action-value function is often described using the Bellman equation, which is presented in Equation (4.6):

$$Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))], \tag{4.6}$$

where E is the environment. Note that the Equation (4.6) assumes that the actor μ is deterministic. This formulation also shows that the Q^μ is only dependent on the environment, which allows off-policy learning of the critic.

To optimize the critics ability to approximate the Equation (4.6) a loss function is needed. Let's denote the exploration policy as β, and the discounted state visitation distribution for a given policy π as ρ^π. Using these notations, the critic's loss-function can be expressed as in the Equation (4.7):

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t|\theta^Q) - y_t)^2], \tag{4.7}$$

where the critic is parameterized by θ^Q. Here $y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$. While y_t depends on the critic parameterization θ^Q, this is often ignored. [19]

To optimize the actor's performance, the gradient of the expected return J is used. The gradient formulation is shown in the Equation (4.8):

$$\nabla_{\theta^\pi} J \approx \mathbb{E}_{s_t \sim \rho^\beta}[\nabla_a Q(s, a|\theta^\pi)|_{s=s_t, a=\pi(s_t)} \nabla_{\theta_\pi} \pi(s|\theta^\pi)|_{s=s_t}]. \tag{4.8}$$

The Equation (4.8) describes the gradient of the policy's performance. It is intuitively clear, that updating the actor to the direction of the policy gradient should improve the policy's performance.[19]

To approximate the functions, neural networks are used. However, many of the optimization methods rely on the assumption that the samples are distributed independently and identically. Clearly, this doesn't hold in reinforcement learning contexts, where the samples are collected during sequential exploration process in a certain environment. To address this problem, the DDPG used a *replay buffer*, similar to the one used in DQN.[19] In this case, the buffer was implemented as a fixed size memory buffer, where s_t, a_t, r_t, s_{t+1} tuples were stored to. When the buffer become full, the old samples were removed in "first in, first out" fashion. To train the agent, the replay buffer was sampled uniformly to create a mini batch of samples, which were used to update the agent. The usage of memory buffer should reduce the effect of sequential sample collection. Note that this replay buffer is not related to the replay buffer mentioned in the Section 4.2.

Another major problem when learning the Q-value function using neural networks is its tendency to diverge. To address this issue, the DDPG introduced target networks $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$. The target networks are initialized to be copies of the actor and critic networks respectively. However, they are not updated directly, and instead the weights of the actor and critic are copied to the targets slowly over time. This updating procedure is called *soft update*, and it is described in Equation (4.9)

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tag{4.9}$$

where θ and θ' are the parameters of the original network and target network respectively. τ is used to control the speed at which the agent learns, and usually $\tau \ll 1$.[19] The soft update improves the stability of the training, but it slows down the convergence.

The TRPO used in the original paper differs from the DDPG used in this work in a few ways. Both, DDPG and TRPO use the policy gradient theorem as the base for their update-rule but use quite different implementations for calculating the update[31], [19]. The TRPO uses an advantage value estimation, and constraint on the update. The constraint is implemented using Kullback-Leiber divergence. This formulation gives TRPO strong theoretical convergence guarantees[31]. As mentioned above, the DDPG uses the actor-critic structure with target networks to optimize the policy, which don't come with similar convergence guarantees. In practice, the DDPG has been more sensitive to the values of the hyperparameters, while the TRPO has achieved almost monotonically increasing performance even in practice[13].

Another major difference between the methods is that DDPG is an off-policy method, whereas the TRPO is an on-policy method. This is a critical difference, as the DDPG can use samples from its replay buffer during learning process. This should make DDPG more sample efficient than TRPO. This is extremely important factor, since in the algorithm 1, the policy is updated during each iteration. If the policy can learn to reach the current set of goals in less steps, it will increase the efficiency of the algorithm significantly, as the improvement affects each iteration. However, it is clear that the possible improvement to the efficiency comes with the cost of stability during training.

4.4 Task

To evaluate the effectiveness of the algorithm, a simple U-shaped maze environment was used, which is shown in the Figure 4.1. The task of the 4-legged spider is to reach the other end of the maze, where the large ball is located. The action-space was 8-dimensional, and each action contained information about the torques applied to the spider's joints. The torques were real numbers in the range of $(-30, 30)$. The observations contained 30 elements, including the velocities and angular velocities of the torso and joints. Additionally, the observation contained the position of the spider's torso. The environment was simulated using the MuJoCo physics simulator[37], and the interaction with the environment was done using the OpenAI Gym[7].

The task was chosen mainly for 2 reasons. Firstly, this task was used in the original work

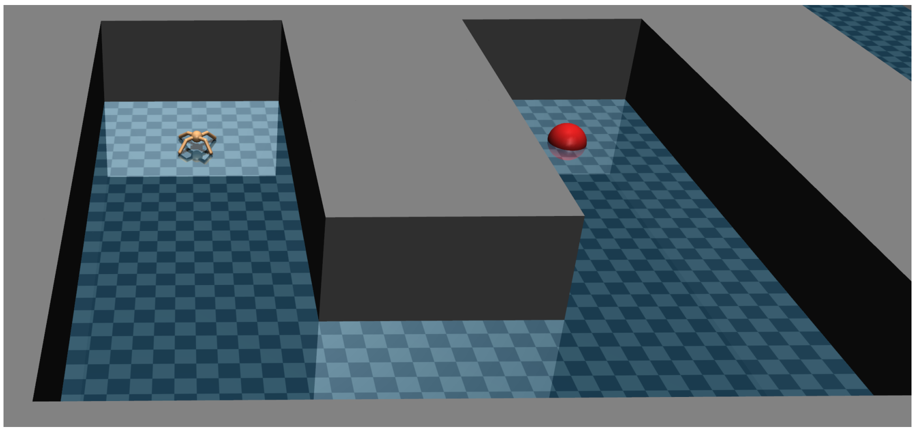


Figure 4.1. *Screenshot of the MuJoCo environment*

describing the Goal-GAN algorithm, and thus their results could be used as guidelines on what can be expected. Secondly, the task of reaching other end of U-shaped maze is extremely simple, yet it is not exactly trivial to define a suitable reward function for this task. A purely distance-based reward function wouldn't work, as the agent must first go around the central wall of the maze, which increases its distance to the goal. Also, the reward function should be able to make distinction between neutral, desirable and undesirable actions. For example, let's suppose that the agent is in the same position as in the Figure 4.1. Now the policy makes an action that moves the agent closer to the central wall of the maze. Should this action be penalized? Technically the agent moved closer to its target, but on the other hand the most desired action would have been a take step forward. This action wasn't unnecessary either, as the agent must move towards the centre of the maze at some point to go around the central wall. Clearly, even in a simple task, it can be quite difficult to define a reward function that signal's clearly to the agent what is the desired way to solve this task, and thus this task clearly illustrates the advantage of using a simple binary reward function, as defined in Equation 4.1.

4.5 Implementation details

In the experiment, during each iteration of algorithm 1, the policy was updated for 5 iterations. Each update iteration consisted of 100 episodes, with each episode having at most 500 time steps. The evaluation of the policy's performance consisted of 100 episodes, where each episode contained at most 500 time steps. In both cases, the task was set to be episodic, i.e. the task and environment were reseted when any of the current goals was found, or when the maximum number of time steps were used.

During each iteration of the algorithm, total of 150 intermediate goals were generated. 100 of these goals were generated by the GAN, and 50 of then were sampled from the replay buffer. The GAN's generator and discriminator contained only 2 linear layers, as the dimensionality of the produced distribution was low. An RMSProp optimizer was used to optimize both the generator and discriminator. The input noise for the generator was 4-dimensional. During each iteration of the algorithm, the GAN was trained for 150 iterations.

The actor and critic of the DDPG were implemented as neural networks as mentioned in the Section 4.3. Both contained 3 linear layers with *ReLU*[24] activation functions between the layers. A *Tanh* activation function was applied to the actor's output, while no activation function was applied to the critic's output. Batch normalization[15] was applied to both actor's and critic's networks. The control parameter τ for the soft update (4.9) was set to 0.001. The replay buffer's capacity was 10 000 samples, and each mini batch sampled from it had 128 samples.

The DDPG is an off-policy method, and thus the exploration was addressed explicitly. In this case, the exploration policy was created by adding noise $\mathcal{N}$ to the actor's current policy, as shown in Equation (4.10)

$$\mu(s_t)' = \mu(s_t|\theta_t^{\pi}) + \mathcal{N}, \tag{4.10}$$

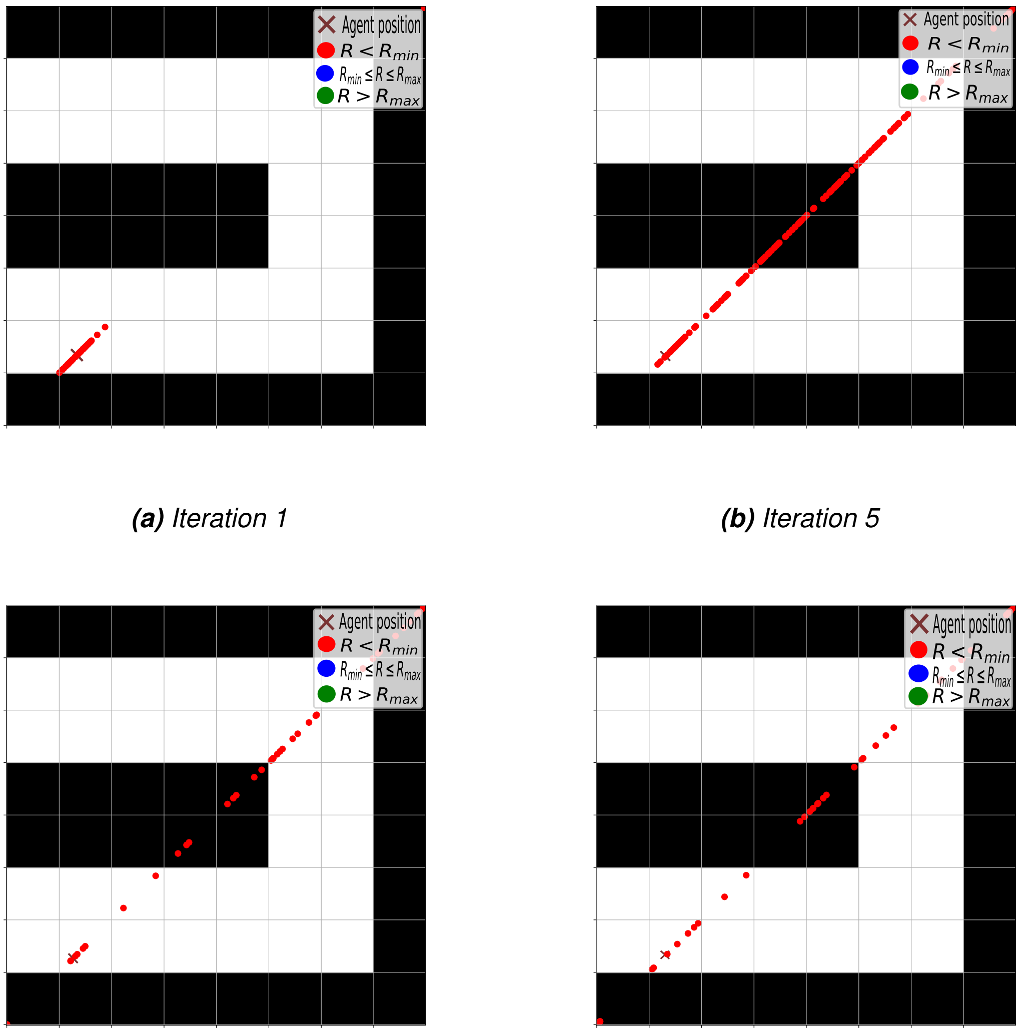
where the noise $\mathcal{N}$ was implemented as Ornstein-Uhlenbeck process[26]. This is same approach as used in the original work introducing the DDPG.

5. RESULTS AND DISCUSSION

In this chapter, the results from the experiment and discussion around them is presented. Lastly, some possible research directions on the subject are discussed. Note that the code to reproduce the results is available at https://github.com/SanteriHei/GoalGAN-DDPG.

The Figure 5.1 shows the intermediate goals that the GAN generated during different iterations of the training process. As can be seen from the Figure 5.1a, the generated intermediate goals are located close to the agent's starting position. This makes sense, as it is quite likely that the initial policy cannot reach further away from the starting position. However, it is also apparent that the initial policy couldn't reach any of the initial goals reliably (all goals are marked with red). The Figures 5.1b, 5.1c and 5.1d show that the generated goals start to move further away from the agent's starting position, even when the agent couldn't reach the intermediate goals that were located closer to it earlier. Additionally, the generated intermediate goals form a diagonal line, which indicates that the GAN used to generate the goals diverged early on during the process. This is somewhat expected, as the GAN received only negative labels from the agent, that is to say, no goal generated by the GAN had correct level of difficulty for the policy's current capabilities. This led to situation, where the GAN couldn't improve the intermediate goals it produced, and thus it ultimately diverged.

In the original paper, Florensa et al. [9] presented results that differ quite significantly from the results presented above. In the original work, the algorithm was tested in the same task as in this study, but in the original work, the authors were able to successfully train the agent. Interestingly, the authors noted that around 20% of the generated goals had sufficient level of difficulty during the whole training process. This result from the original experiment indicates two things: Firstly, the original work demonstrated that the GAN can adjust the difficulty level of the goals it was producing based on the agent's performance. Secondly, the training process seems to be quite stable, if the agent is able learn to reach the goals during the first few iterations. This underlines the importance of the first few iterations of the algorithm. If the agent cannot learn during the first few iterations, the algorithm will diverge quickly, as shown in this study. However, if it can learn during the first few iterations, then it seems likely that the algorithm can train the agent successfully, as the original paper described.



***(a)** Iteration 1*

***(b)** Iteration 5*

***(c)** Iteration 20*

***(d)** Iteration 99*

***Figure 5.1.** Goals generated during different iterations*

The policy's inability to reach even the easiest goals reliably can be explained by the fact that it didn't receive enough rewards during the training. This is a known problem when working with binary reward functions, such as one defined in Equation (4.1). Figure 5.2 shows the average amount of goals reached in the policy's evaluation phase during each iteration of the algorithm. It is clear from the Figure 5.2 that the agent did not receive lot of rewards during the evaluation. The DDPG's update relies on the fact that the gradient of the policy's performance will provide a direction which improves the performance. However, if the agent doesn't receive enough rewards, as was case in the experiment, the gradient cannot provide a direction that improves the performance of the policy. In extreme case, where the agent receives only single reward values (0's in this case), the gradient is taken over a flat surface, which cannot provide any meaningful information on how to improve the performance of the policy.

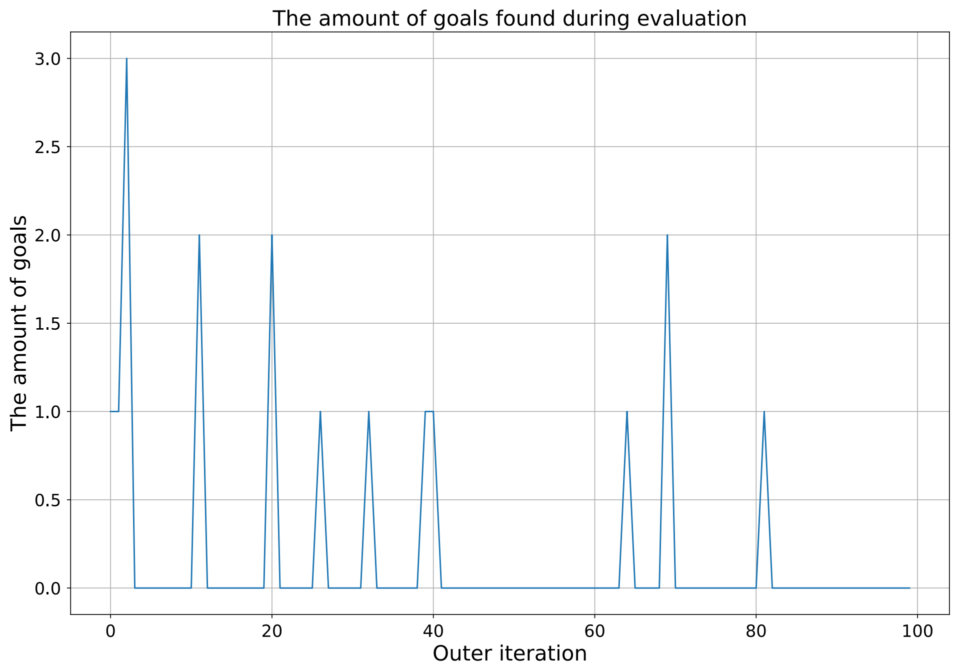


Figure 5.2. *The average amount of goals found during evaluation of the policy*

Now it is important to understand why the policy couldn't reach the initially reached goals consistently enough. The Figure 5.2 shows that the policy could reach at most 3 goals on average during the evaluations, after which its performance decreases rapidly. This is most likely due to insufficient exploration strategy. It is clear from Figure 5.1a, that the initially generated goals were located so close to the agent's starting position that the policy should have reached them more consistently. However, it seems that the exploration strategy explained in Section 4.5 wasn't robust enough, which meant that event the easiest goals couldn't be reached. This in turn lead to suboptimal updates to the policy, which in turn leads to the policy's divergence. In practice, the actions produced by the policy saturated to the limits of the action space.

Additionally, the algorithm's initialization mechanism isn't very robust. Firstly, it relies on the agent's exploration capabilities, rather than providing more robust and general solution. Secondly, even if the agent has good exploration capabilities, the initialization still relies most likely on some kind of random walk procedure to produce a good estimate on the policy's initial performance. This is both unreliable and expensive, as it requires extensive number of steps in the environment.

The insufficient exploration capabilities in addition to the problems with the initialization procedure led to the situation where the policy could not learn to reach even the initial intermediate goals consistently. The experiment and the analysis presented above make it clear that the few initial iterations are extremely crucial for the algorithm's performance.

During this experiment, the later iterations where the policy could reach some goals are quite arbitrary, and don't provide any meaningful insights, as the policy's actions saturated early on during the training.

Based on the experiment, it is clear that further work is required for applying Goal-GAN algorithm to off-policy methods. As mentioned above, a more robust initialization algorithm could benefit the algorithm. One possible solution could be to just assume that the policy's initial reach is quite limited and sample the initial goals uniformly from certain radius from the agent's starting point. This would make the initialization computationally cheaper but defining the feasible radius would be another environment dependent variable to control. Also, a hybrid solution could be used, where some of the initial goals are sampled uniformly from are close to the agent's starting position, while some of them would be generated based on the states that the agent visited during the exploration. This could be used to reduce the number of steps done in the environment, while still providing a more accurate guess on the goals that the policy can reach.

One interesting possibility would be to combine Hindsight Experience Relay (HER)[1] with the Goal-GAN algorithm. The HER is used to store trajectories that the agent has explored during previous episodes. Those trajectories are then replayed, but with different goal. The problem with HER is that one must choose a heuristic that is used to select the new goal. The GAN used in the Goal-GAN algorithm could be used to propose a new goal for the replayed trajectory.

Another possible research topic for the future could be the usage of other simple reward functions in place of the binary indicator function. This would somewhat reduce the generality of the method, but in certain cases a more specific reward function could produce better results. For example, in the experiment a negative distance function could have been used to provide the policy with dense reward, without defeating the purpose of the algorithm. One could still avoid the need to carefully design a single purpose reward function, and rather use a simple and easy-to-understand reward function. On this note, a few experiments were done using the negative distance as the reward function, but the results were similar to the ones presented above. The change of reward function would require extensive rework of the algorithm, as the semantics of the reward function would change significantly. As mentioned in Chapter 4, currently the reward function can be interpreted as a probability of reaching a given goal g in a T time steps. However, this wouldn't be true if the reward function would be changed. For example, when using the negative distance as the reward function, one must consider following: How exactly the reward is calculated? Is it a mean of the distances to each of the goals? Or is it minimum/maximum of those distances? The answer to the above questions isn't exactly clear, and each of the options come with its own benefits and drawbacks. The minimum and maximum use information about one goal, which raises the question that is it necessary to generate set of goals? On the other hand, the mean of the distances doesn't lead the agent to any particular goal, which might confuse the agent. Either way, the change of reward function would require significant work around the interpretation of the sum of individual reward functions.

It is also important to note that the task used in the experiment doesn't provide a comprehensive look to the performance of the algorithm. The used task was quite simple, and notably it had no complex interactions with the environment. The agent doesn't have to carefully avoid obstacles that it could knock over, nor does it have to interact with any obstacles in any other ways (e.g. grasping an object). Additionally, the goal-space of the task is simple, and it doesn't assess the GAN's ability to model complex high dimensional goal-spaces.

Lastly, the author is not aware that any attempts have been made to apply the algorithm to real-world problems outside of the simulator. This is direction that should be researched in the future, but more work is required to ensure the safety of the agent's actions during the training. It must be noted that the algorithm's requirement to be able to reset the environment at any time might limit the applications where the algorithm can be used.

One big shortcoming of the algorithm is its lacklustre performance. It is easy to see from algorithm 1, that the procedure contains many nested loops, which makes the algorithm extremely time consuming. The training of the policy is an element that cannot be changed too much. However, the evaluation of the policy's performance might offer possibilities for performance optimizations. Currently, the policy's performance is evaluated during each iteration by letting the policy to complete the given intermediate goals. More applicable approach could be to approximate the policy's performance, rather than testing it directly. In certain cases, the training performance of the policy could be used as indicator of the policy's real performance, but this is dependent on how similar the exploration policy is to the policy used to complete the tasks. The performance of the algorithm is an open question, which would require more thorough research.

In the end, it is clear that producing curricula automatically, while preserving the generality of the algorithm is not ready to replace reward function shaping. However, the promise of removing the need for careful reward function shaping is extremely tempting. The experiment showed the initialization procedure and exploration design are problems that must be solved first. The algorithms scalability to more complex environments and its suitability to real-world situations are open questions that should be researched in the future.

REFERENCES

[1] Marcin Andrychowicz et al. *Hindsight Experience Replay*. 2017. DOI: 10.48550/ARXIV.1707.01495. URL: https://arxiv.org/abs/1707.01495.

[2] Martin Arjovsky and Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. arXiv: 1701.04862 [stat.ML].

[3] Adrien Baranes and Pierre-Yves Oudeyer. "Active learning of inverse models with intrinsically motivated goal exploration in robots". eng. In: *Robotics and autonomous systems* 61.1 (2013), pp. 49–73. ISSN: 0921-8890.

[4] Andrew G Barto, Satinder Singh, Nuttapong Chentanez, et al. "Intrinsically motivated learning of hierarchical collections of skills". In: *Proceedings of the 3rd International Conference on Development and Learning*. Piscataway, NJ. 2004, pp. 112–19.

[5] Yoshua Bengio et al. "Curriculum Learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 41–48. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380. URL: https://doi-org.libproxy.tuni.fi/10.1145/1553374.1553380.

[6] Nicolas Bougie and Ryutaro Ichise. "Skill-based curiosity for intrinsically motivated reinforcement learning". eng. In: *Machine learning* 109.3 (2019), pp. 493–512. ISSN: 0885-6125.

[7] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[8] Manfred Eppe, Sven Magg, and Stefan Wermter. "Curriculum goal masking for continuous deep reinforcement learning". In: *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. 2019, pp. 183–188. DOI: 10.1109/DEVLRN.2019.8850721.

[9] Carlos Florensa et al. "Automatic Goal Generation for Reinforcement Learning Agents". eng. In: (2017).

[10] Carlos Florensa et al. *Reverse Curriculum Generation for Reinforcement Learning*. 2018. arXiv: 1707.05300 [cs.AI].

[11] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[12] Zhenghua He et al. "Automatic Curriculum Generation by Hierarchical Reinforcement Learning". eng. In: *Neural Information Processing*. Vol. 12533. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 202–213. ISBN: 9783030638320.

[13] Peter Henderson et al. *Deep Reinforcement Learning that Matters*. 2017. DOI: 10.48550/ARXIV.1709.06560. URL: https://arxiv.org/abs/1709.06560.

[14] Jiang Hua et al. "Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning". eng. In: *Sensors (Basel, Switzerland)* 21.4 (2021), pp. 1–21. ISSN: 1424-8220.

[15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/ARXIV.1502.03167. URL: https://arxiv.org/abs/1502.03167.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement Learning: A Survey". English. In: *The Journal of Artificial Intelligence Research* 4 (1996). Copyright - © 1996. Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the associated terms available at https://www.jair.org/index.php/jair/about; Last updated - 2021-07-23, pp. 237–285. URL: https://libproxy.tuni.fi/login?url=https://www.proquest.com/scholarly-journals/reinforcement-learning-survey/docview/2554154256/se-2.

[17] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". eng. In: *The International journal of robotics research* 32.11 (2013), pp. 1238–1274. ISSN: 0278-3649.

[18] Ruijia Li et al. "Anchor: The achieved goal to replace the subgoal for hierarchical reinforcement learning". In: *Knowledge-Based Systems* 225 (2021), p. 107128. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.107128. URL: https://www.sciencedirect.com/science/article/pii/S0950705121003919.

[19] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. DOI: 10.48550/ARXIV.1509.02971. URL: https://arxiv.org/abs/1509.02971.

[20] A. Rupam Mahmood et al. *Benchmarking Reinforcement Learning Algorithms on Real-World Robots*. 2018. DOI: 10.48550/ARXIV.1809.07731. URL: https://arxiv.org/abs/1809.07731.

[21] Xudong Mao et al. *Least Squares Generative Adversarial Networks*. 2017. arXiv: 1611.04076 [cs.CV].

[22] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. DOI: 10.48550/ARXIV.1312.5602. URL: https://arxiv.org/abs/1312.5602.

[23] Eduardo F Morales et al. "A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning". eng. In: *Intelligent service robotics* 14.5 (2021), pp. 773–805. ISSN: 1861-2776.

[24] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve Restricted Boltzmann machines". eng. In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*. 2010, pp. 807–814. ISBN: 9781605589077.

[25] OpenAI et al. *Learning Dexterous In-Hand Manipulation*. 2018. DOI: 10.48550/ARXIV.1808.00177. URL: https://arxiv.org/abs/1808.00177.

[26] Leonard Salomon Ornstein. “On the theory of the Brownian motion”. eng. In: *Physical review* 36.5 (1930), pp. 823–841. ISSN: 0031-899X.

[27] Parag C Pendharkar and Patrick Cusatis. “Trading financial indices with reinforcement learning agents”. eng. In: *Expert systems with applications* 103 (2018), pp. 1–13. ISSN: 0957-4174.

[28] Roger Ratcliff. “Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions”. eng. In: *Psychological review* 97.2 (1990), pp. 285–308. ISSN: 0033-295X.

[29] Lars Ruthotto and Eldad Haber. “An introduction to deep generative modeling”. eng. In: *Mitteilungen der Gesellschaft für Angewandte Mathematik und Mechanik* 44.2 (2021). ISSN: 0936-7195.

[30] Divya Saxena and Jiannong Cao. “Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions”. In: *ACM Comput. Surv.* 54.3 (May 2021). ISSN: 0360-0300. DOI: 10.1145/3446374. URL: https://doi-org.libproxy.tuni.fi/10.1145/3446374.

[31] John Schulman et al. *Trust Region Policy Optimization*. 2015. DOI: 10.48550/ARXIV.1502.05477. URL: https://arxiv.org/abs/1502.05477.

[32] David Silver et al. “Deterministic Policy Gradient Algorithms”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, June 22–24, 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html.

[33] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. eng. In: *Nature (London)* 529.7587 (2016), pp. 484–489. ISSN: 0028-0836.

[34] Yeeho Song and Jeff Schneider. *Robust Reinforcement Learning via Genetic Curriculum*. 2022. DOI: 10.48550/ARXIV.2202.08393. URL: https://arxiv.org/abs/2202.08393.

[35] Sainbayar Sukhbaatar et al. *Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play*. 2017. DOI: 10.48550/ARXIV.1703.05407. URL: https://arxiv.org/abs/1703.05407.

[36] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[37] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.