ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

GAN-BASED INTRINSIC EXPLORATION FOR SAMPLE EFFICIENT REINFORCEMENT LEARNING

M.Sc. THESIS

Doğay KAMAR

Department of Computer Engineering

Computer Engineering Programme

MAY 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

GAN-BASED INTRINSIC EXPLORATION FOR SAMPLE EFFICIENT REINFORCEMENT LEARNING

M.Sc. THESIS

Doğay KAMAR
(504181511)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Prof. Dr. Gözde Ünal

MAY 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ÖRNEK VERİMLİ PEKİŞTİRMELİ ÖĞRENME İÇİN ÜRETKEN ÇEKİŞMELİ AĞLARLA İÇSEL KEŞİF

YÜKSEK LİSANS TEZİ

Doğay KAMAR
(504181511)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Gözde Ünal

MAYIS 2022

Doğay KAMAR, a M.Sc. student of ITU Graduate School student ID 504181511 successfully defended the thesis entitled "GAN-BASED INTRINSIC EXPLORATION FOR SAMPLE EFFICIENT REINFORCEMENT LEARNING", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Gözde Ünal** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Nazım Kemal Üre** Istanbul Technical University	.....
	Prof. Dr. Yücel Yemez Koç University	.....
		.....

Date of Submission : **10 May 2022**
Date of Defense : **23 May 2022**

To my loved ones,

FOREWORD

I hadn't realized that I wanted to do academic research until I finished my bachelor's degree. When I decided to study artificial intelligence, I was starting my master's degree and I felt I lacked the necessary prior knowledge and experience compared to my peers. I would like to thank Dr. Damien Jade Duff for helping me take my first steps into this field as smoothly as possible, having never-ending patience with me, showing me how fun and joyful learning can be and his guidance that shed light on my path. He might not have realized, but I am where I am thanks to Dr. Duff.

I would like to thank Prof. Dr. Gözde Ünal and Assoc. Prof. Nazım Kemal Üre for their continuous guidance throughout the studies of this thesis. They helped tremendously through the first published work that led to this thesis. Also I would like to thank my family and friends for always being there for me when I needed it.

Without any of the mentioned individuals above, this thesis wouldn't be.

May 2022

Doğay KAMAR
Computer Engineer

TABLE OF CONTENTS

ABBREVIATIONS

GAN	**:** Generative Adversarial Network
GIRM	**:** GAN-Based Intrinsic Reward Module
MDP	**:** Markov Decision Process
DORA	**:** Directed Outreaching Reinforcement Action-Selection
RND	**:** Random Network Distillation
NGU	**:** Never Give Up
DTSIL	**:** Diverse Trajectory-conditioned Self Imitation Learning
ANN	**:** Artificial Neural Network
CNN	**:** Convolutional Neural Network
WGAN	**:** Wasserstain GAN
A2C	**:** Advantage Actor-Critic
A3C	**:** Asynchronous Advantage Actor-Critic

SYMBOLS

ε	: The probability of choosing a random action over selected action.
G	: Generator
D	: Discriminator
s_t	: State at time step t
a_t	: Action at time step t
r_t	: Reward at time step t
r_t^i	: Intrinsic reward at time step t
r_t^e	: Extrinsic reward at time step t
$\mathbb{M}$	: Markov Decision Process
$\mathbb{S}$	: Set of possible states
$\mathbb{A}$	: Set of actions
$\mathbb{R}$	: Reward function
$\mathbb{P}$	: State transition probability distribution
ρ_0	: Distribution of the initial state
π	: Policy of the agent
π^*	: Optimal policy
γ	: Discount factor
$V(s)$	: Value of a state
$\boldsymbol{z}$	: Sampled noise input
$p_{\boldsymbol{z}}$	: Input noise distribution
$\boldsymbol{x}$	: Real data sample
E	: Encoder
$\mathscr{L}(s_t)$	: MSE Loss of Encoder
$\mathscr{L}_D(s_t)$	: Discriminator Loss of Encoder
λ	: Weight of the discriminator loss
n	: Number of dimensions in the state space
f	: Output of an intermediate layer in discriminator
n_d	: Dimension of the intermediate layer output
EMA	: Exponentially moving average
EMV	: Exponentially moving variance
M	: State memory
N	: Size of the state memory
$A(s_t, a_t)$	: Advantage of an action

LIST OF TABLES

Page

LIST OF FIGURES

Page

GAN-BASED INTRINSIC EXPLORATION FOR SAMPLE EFFICIENT REINFORCEMENT LEARNING

SUMMARY

Reinforcement learning is a sub-area of artificial intelligence in which the learner learns in a trial-and-error manner. The learner does so by executing an action depending on the current state it is in and observing the results. After executing an action, a reward signal is given to the learner and through the rewards, the learner can learn which actions are best in different situations. However, the learner is not given any prior information about the environment it is in or which action is the best depending on the current state. Therefore, exploring the environment is important for gathering the necessary information in order to navigate to the high rewards.

Most common exploration strategies involve random action selection occasionally. However, they only work under some conditions such that the rewards need to be dense and well-defined. These conditions are hard to meet for many real-world problems and an efficient exploration strategy is needed for such problems.

Utilizing the Generative Adversarial Networks (GAN), this thesis proposes a novel module for sample efficient exploration, called GAN-based Intrinsic Reward Module (GIRM). The GIRM computes an intrinsic reward for the states and the aim is to compute higher rewards for the novel, unexplored states. The GIRM uses GAN to learn the distribution of the states the learner observes and contains an encoder, which maps a query state to the input space of the generator of the GAN. Using the encoder and the generator, the GIRM can detect if a query state is among the distribution of the observed states. If it is, the state is regarded as a visited state, otherwise, it is a novel state to the learner, in which case the intrinsic reward will be higher. As the learner receives higher rewards for such states, it is incentivized to explore the unknown, leading to sample-efficient exploration.

The GIRM is evaluated using two settings: a sparse reward and a no-reward environments. It is shown that the GIRM is indeed capable of exploring compared to the base algorithms, which involve random exploration methods, in both of the settings. Compared to the other studies in the field, the GIRM also manages to explore more efficiently in terms of the number of samples. Finally, we identify a few weaknesses of GIRM: the negative impact on the performance when sudden changes to the distribution of the observed states occur, and the exploitation of very large rewards not being avoided.

ÖRNEK VERİMLİ PEKİŞTİRMELİ ÖĞRENME İÇİN ÜRETKEN ÇEKİŞMELİ AĞLARLA İÇSEL KEŞİF

ÖZET

Pekiştirmeli öğrenme, öğrenmenin deneme ve yanılma metoduyla gerçekleştiği, yapay zekanın bir alt alanıdır. Pekiştirmeli öğrenmede bir öğrenici, veya etmen, içinde bulunduğu ortam ile, her zaman adımında etkileşimde bulunur. Etmenin amacı, bu ortam içerisinde tanımlanan problemi çözmektir. Etmen bunu, her zaman adımında, problemin bulunduğu ortamda, o anki durumun gözlemine göre, yapabileceği aksiyonların kümesinden bir aksiyon seçerek ve bu aksiyonu uygulayarak yapmaktadır. Ortam ise, etmenin seçtiği ve uyguladığı aksiyonun sonucu olarak etmene bir ödül sinyali gönderir. Etmenin amacı, tek bir döngüde, ortamdan topladığı ödülleri olabildiğince yüksek yapacak şekilde aksiyonlar seçmektir. Ancak etmen, öğrenme öncesinde, ortamla ilgili veya aksiyonların ne yaptıklarıyla veya hangi aksiyonun hangi durumda daha iyi olduğuna dair herhangi bir bilgiye sahip olmaz. Etmen, ortamla etkileşime girerek aldığı ödüller sayesinde aksiyonlarının sonucunu öğrenmektedir. Bunu yapabilmesi için, etmenin ortamla ilgili gerekli bilgiye sahip olması amacıyla, ortamı keşfetmesi gerekir. Keşif yaptığı zaman adımlarında etmen, o ana kadar sahip olduğu bilgiye rağmen, yalnızca ortamı keşfetme amacıyla bir aksiyon seçer. Keşif süreci, etmenin belli bir bilgiye sahip olmasıyla sonlanmayabilir, zira etmen bir çözüm bulsa bile, ortamda bulduğundan daha iyi bir çözüm bulunabilir ve o ana kadar bulduğu çözümü sömürmek yerine, daha iyi bir çözüm aramak için de keşif amacıyla aksiyonlar seçilmesi mümkündür.

En yaygın kullanılan keşfetme stratejileri, rastgele bir şekilde aksiyon seçmeye dayalılardır. Bu stratejiler, belli koşullar altında bazı zaman aralıklarında, etmenin en iyi aksiyon olarak seçtiği aksiyonu göz ardı ederek, rastgele aksiyon seçimine yönlendirecek bir metotu takip ederler. Bu rastgelelik sayesinde ise etmen, ortamda daha önce keşfedemediği durumları gözlemleyebilir ve yeni bilgiler edinebilir. Geçmişte bu metotlar, birçok pekiştirmeli öğrenme probleminde başarılı bir performans göstermiş ve dolayısıyla yaygınlaşmışlardır. Hala yaygın olarak kullanılan bu metotlar, günümüzdeki çoğu pekiştirmeli öğrenme problemlerinde yeterli keşfetmeyi sağlamaktadır. Ancak rastgele seçime bağlı çalışan bu metotların işe yaramaları için bazı şartların yerine gelmesi gerekmektedir. Öncelikle ödül sinyallerinin sık olması, devamında da ödüllerin iyi bir şekilde tanımlanmış olmaları, yani iyi aksiyonları kötü aksiyonlardan doğru bir skala ile ayırt edebilmesi gerekmektedir. Gerçek hayattaki birçok problem ise bu şartları her zaman sağlayamamaktadır. Özellikle seyrek ödüllerin olduğu veya hedef noktası dışında herhangi bir ödülün olmadığı problemler mevcuttur. Bu tarz problemlerde rastgele aksiyon seçimiyle ortamdan herhangi bir ödül bulmak zordur, dolayısıyla etmeni eğitecek herhangi bir geribildirimde bulunamamaktadır. Bu nedenle, daha etkili bir bir keşfetme stratejisinin geliştirilmesi

gerekmektedir. Etkili keşif, uzun zamandır araştırma konusu olmuş ve birçok çalışma bu konuyu ele almıştır. Günümüzde halen kesin bir çözüme ulaşılamamış ve ucu açık bir çalışma alanı olarak yer almaktadır.

Bu tezin konusu, efektif keşfetmeye yeni bir çözüm önerisi getirmek üzerinedir. Tezde, Üretken Çekişmeli Ağlar (ÜÇA) kullanılarak, ÜÇA-temelli İçsel Ödül Modülü (ÜİÖM) adını verdiğimiz, özgün bir örnekçe efektif, etmeni keşif yapmaya teşvik eden bir modül önerilmiştir. ÜİOM'un pekiştirmeli öğrenme algoritmalarındaki görevi, her zaman adımındaki durum için, ortamdan gelen dışsal ödül sinyaline ek olarak bir içsel ödül hesaplamaktır. İçsel ödüllerin ise, yeni, daha önce etmen tarafından ziyaret edilmemiş durumlar için yüksek olması, sıklıkla görülmüş ve etmen tarafından bilinen durumlar içinse düşük olması için ÜİOM eğitilmektedir. Yeni durumların yüksek ödüllü olmaları, etmeni bu durumları daha sıklıkla ziyaret etmeye teşvik etmekte, dolayısıyla daha efektif bir keşif sağlamaktadır. Ayrıca ödülün olmadığı veya seyrek olduğu ortamlarda da, etmene içsel ödüller aracılığıyla bir geri besleme yapılabilmekte, etmenin ortamdaki hedefi veya seyrek ödülleri bulana kadar efektif keşif yapması sağlanmaktadır.

ÜİÖM içerisinde ÜÇA'nın yanı sıra bir de kodlayıcı model bulunur. ÜÇA'nın kullanım amacı, etmenin gözlemlediği durumların dağılımını öğrenmektir. Kodlayıcı ise, sorgu durumunun, ÜÇA'daki üretici ağın giriş vektör uzayına haritalandırılması amacıyla kullanılır. Bu iki varlık, ÜÇA ve kodlayıcı, bir anomali tespit etme görevinde olduğu gibi çalışarak, ziyaret edilmemiş durumları tespit etmeye çalışır. Kodlayıcı, bir sorgu durumunu önce üreticinin giriş vektör uzayına haritalandırır, çıktı vektörü ise ÜÇA'daki üretici ağ ile tekrar oluşturulur. Bu oluşturulan yeni durumun, sorgu durumuna benzerliği kullanılarak içsel ödül hesaplanır. Eğer benzerlik fazlaysa, bunun anlamı sorgu durumu, ÜÇA tarafından öğrenilen dağılımda yer alan bir durumdur, yani etmen tarafından gözlemlenmiştir. Bu durumda düşük bir içsel ödül verilir. Benzerliğin az olması ise, ÜÇA'nın sorgu durumunu yeniden oluşturamaması, yani bu sorgu durumunu daha önce görmemiş olması anlamına gelir. Bu durumda ise yeni bir durumun gözlemlendiği tespit edilir ve yüksek bir içsel ödül verilir. İçsel ödülün yeni durumlarda yüksek olması, etmeni bu durumları daha çok ziyaret etmeye itmektedir. ÜİÖM'ün eğitimi periyodik olarak tekrarlanmasıyla da, gözlemlenen durumların dağılımı güncel tutulur ve içsel ödülün devamlı olarak tutarlı bir şekilde keşfe teşvik etmesi sağlanır. Bu periyodik eğitim, bir durum belleğiyle sağlanmaktadır. Belirli kapasitedeki bellek, gözlemlenen durumları saklar ve bu belleğin dolduğu durumda ÜİÖM, bellekteki verilerle eğitilir. Her eğitim sonunda bellek boşaltılır ve yeni gözlemlenen durumlar toplanmaya başlanır. Her belleğin doluşunda, yeni gözlemlenen veriler ile eğitim sağlandığı için, ÜİÖM'ün gözlemlenen verilere adapte olması sağlanır. Ek olarak, ÜİÖM, pekiştirmeli öğrenme sırasında periyodik olarak tekrar eğitildiği için, hesaplanan ödüllerin de skalasının şiddetli, performansı etkileyecek şekilde değişmemesi için, içsel ödüllerin standartlaştırılması da ek olarak sağlanmaktadır. Bu sayede içsel ödüller, pekiştirmeli öğrenme boyunca benzer skalada tutulacak ve yeni durumların aldığı içsel ödüller benzer olacaktır. ÜİÖM, bir ek modül olarak tasarlanmıştır, dolayısıyla herhangi bir pekiştirmeli öğrenme algoritması ile beraber kullanılabilmektedir. Bu çalışmada Avantajlı Aktör Kritik algoritması ile kullanılmış, karşılaştırmada da bu algoritmanın orijinal keşif metodu kullanılmıştır.

ÜİÖM'ün değerlendirilmesi iki farklı ortam düzeninde yapılmıştır. Bunlardan birincisi, seyrek ödüllü düzen, ikincisi ise ödülsüz düzendir. Ödülsüz düzen için Super Mario Bros. ortamındaki tüm ödüller kaldırılmış ve etmen bu ortamda eğitilmiştir. Bu ortamda etmenin yeni durumlar keşfetmesi için haritadaki engellere takılmadan sağa doğru hareket etmesi gerekmektedir. Seyrek ödüllü düzen içinse, keşfin zorluğundan ötürü çözülmesinin zorluğuyla bilinen Montezuma's Revenge ortamında yapılmıştır. Bu ortamda ilk ödülü ve devamındaki ödülü bulabilmesi için, etmenin spesifik bir aksiyon sıralamasını yapması gerekmektedir. Bu sıralama ise rastgele bulmak için fazla uzundur ve bu sıralama sırasındaki hataların oyun içerisinde ölümle sonuçlanması ve başa dönülmesi çok muhtemeldir. Bu ortamlarda, ÜİÖM'ün, değerlendirme için rastgele aksiyon seçmeye dayalı keşif yöntemi kullanan baz algoritmalarlar karşılaştırıldığında, etkili bir şekilde keşif yapabildiği gösterilmiştir. Alandaki diğer çalışmalarla karşılaştırıldığında ise, ÜİÖM'ün örnekçe efektif olduğu, diğer çalışmalara göre daha az örnekle sonuç alınabildiği gösterilmiştir. Montezuma's Revenge ortamında çözüme ulaşılamamış veya diğer çalışmalar geçilmemiş olsa bile, farkla daha az örnek ile sonuca ulaşılabildiği görülmektedir. Son olarak, ÜİÖM'ün bazı zayıflıkları da belirtilmiştir. Çok büyük ödül veren bir durumla karşılaşıldığında, ÜİÖM, pekiştirmeli algoritmaların ödül sömürmeye yatkınlığından kaçınamamıştır. Ek olarak, gözlemlenen durumların dağılımındaki ani değişikliklerin ÜİÖM'ün performansını olumsuz şekilde etkilediği gözlemlenmiştir. Tezin sonunda, ÜÇA'ların pekiştirmeli öğrenme ile karşılıklı olarak birçok konuda geliştirmeler yapılabileceği ve bu çalışmanın bir önayak olmasının umulduğu belirtilmiştir.

1. INTRODUCTION

This thesis presents a novel solution for sample-efficient exploration in reinforcement learning using Generative Adversarial Networks(GAN) [9].

1.1 Reinforcement Learning and The Exploration-Exploitation Trade-off

Reinforcement learning is a branch of Artificial Intelligence, in which learning occurs through experiences. In a reinforcement learning setup, an agent is trained to learn to choose the best action to take from an action set, depending on the observation of the state in the environment it is in. Every time the agent takes an action, it is provided a reward signal by the environment, and the agent's goal is to maximize the cumulative reward it receives through the action selection policy it has learned. (Sutton and Barto, 2018). In most setups, the agent does not know any prior about the environment and it is not given any clue about which action is the best as most problems in real life do not provide such information. Therefore, the agent needs to learn about the environment to find the best action policy. To do so, reinforcement learning algorithms often incentivize the agent to select actions to explore the environment instead of exploiting its current knowledge and trying to select actions that maximize the cumulative reward. The former and latter action selection strategies usually contradict each other, creating the dilemma known as "the exploration-exploitation trade-off". Exploring unnecessarily when necessary information has already been gathered leads to the agent receiving less rewards while exploiting the knowledge without exploring sufficiently leads to non-optimal solutions found by the agent. Exploitation can be achieved by simply selecting the action that returns the best value, i.e. maximizes the cumulative reward from the current state of the environment using the knowledge the agent has gathered. However, efficient exploration can not be formulated as simply due to the problem being exploring the unknown and still remains an open problem in

the field.

1.2 The Efficient Exploration Problem in Reinforcement Learning

Exploration is a crucial concept in the reinforcement learning that is required to gather necessary data for the agent. The importance of the exploration is due to the fact that the agent does not have any prior information about the environment, which means any state the agent observes at the beginning is novel to itself and it does not know the dynamics of the possible actions, i.e. what each action does. The agent learns which states are desired, and which actions are the best depending on the observed state, by the reward signals it receives from the environment. The ideal way for training the agent to adapt the best action selection policy to maximize the cumulative reward would be to visit as many states as possible and learn through the results of the state-action pairings for each state it visits during exploration. However, this is not a feasible strategy due to two reasons: firstly, it is a difficult task to devise a generalized method that can visit all the states which can work on all environments in a sample-efficient manner, and secondly, and more importantly, many modern problems feature either infinitely or near-infinitely big state space, which renders observing all states impossible. Moreover, as opposed to humans, the agent needs to receive a reinforcement several times for a single instance of an interaction, requiring the agent to observe the same state multiple times.

Most common exploration approaches, such as ε-greedy [10], adding noise to action [11], maximizing entropy [12] etc. depend on a random process for action selection instead of using the agent's decision making in order to increase the variance of the selected actions for observed states. Coupled with the aforementioned problem of traversing the state space, random exploration approaches can only work if they receive a reward signal from the environment continuously, i.e. a reward is received at almost every time step, and if the reward function can correctly return positive rewards for the actions that lead to the goal and negative rewards for the unintended behavior that are meant to be penalized. The reward function should also be able to assess the importance of the positive and negative actions; the reward received for reaching the

goal state should be higher than the reward for approaching the goal. Likewise, the penalty for completely failing the task should also be significantly higher than the penalty for a minor step-back away from the goal. With such a reward function, from its limited collected data, the agent can learn a generalized action policy that works for the whole environment or it can reach the later stages to collect new data to learn from. However, most real-world problems come with an infinitely big state-space, and the rewards received from the environment are extremely sparse, if there are any rewards at all. In such problems, with random exploration approaches, the observed parts of the environment by the agent is limited to a small percentage of the state space that are mostly around the initial state as it depends on only the randomness of the selected action, and to perform the action sequence that leads to the goal state by just selecting random action is impossible. Therefore, in environments with sparse or no rewards, an additional reinforcement for efficient exploration is important.

1.3 Motivation

To solve the efficient exploration problem, most of the recent studies [3,4,13]–[19] have focused on proposing modules that compute an additional reward, called "intrinsic reward", in addition to the reward that is received from the environment, which is also called "extrinsic reward". Following this trend, we propose a module that incorporates Generative Adversarial Networks(GAN) [9] with reinforcement learning, called "GAN-Based Intrinsic Reward Module"(GIRM), which is also tasked to provide an intrinsic reward to the agent. GANs are composed of two models that are adversarial to each other: a generator G tasked with generating synthetic data similar to real data given a noise input sampled from a pre-defined distribution, and a discriminator D tasked with predicting whether the input data it receives is real or generated. These two models are tasked adversarially against each other: G tries to fool D that the data it generates is real, while D aims to distinguish between real data and the data G generates, detecting the generated data. For GAN to converge as intended, both G and D needs to be trained well, as the improvement of the performance of these models depends on the success of each other. In an ideal training, G can generate data that is hard to distinguish from real data, and it does so by learning and fitting to the

distribution of the real data, thus being able to sample a new, generated data from the distribution it learned. This leads to the idea that, if the right noise input is given, G can also generate an output that is the same as a sample from the original data. This idea is realized for an anomaly detection problem in the works of Schlegl et al. [20] and Schlegl et al. [21], in which, after training the GAN, G is used to generate the same sample as the query sample, and the difference between the query and the generated sample is assigned as an anomaly score. If the measured anomaly score is higher than a threshold, it is treated as an anomaly. Inspired by the usage of GAN in such a task [21], our proposed module GIRM is trained on the visited state observations, similar to the GANs being trained with real data, and the trained G is used to compute an intrinsic reward in addition to the extrinsic reward, and the unobserved, novel states are assigned higher rewards due to them not being in the data used for training of the GIRM, thus leading to the higher difference between generated and observed state data. As a result, the GIRM leads to more sample-efficient exploration in a reinforcement learning algorithm, by incentivizing the agent to visit the novel states as they are assigned higher intrinsic rewards. Our motivation behind using GANS for this task is (a) the GIRM is only trained to learn the distribution of the observed states and is not required to learn the dynamics of the environment, leading to less complex requirements and (b) GANS are very widely studied and its use-cases have expanded to many different study fields. We believe that GANs and reinforcement learning show similarities and by integrating one into another, we showcase a successful application we hope can open the gates for future studies.

2. BACKGROUND AND LITERATURE REVIEW

This chapter presents the necessary preliminaries in order to understand the method presented in this thesis and the motivation behind it. First, we explain the basics of reinforcement learning, how the optimal policy is formulated and the necessity of the exploration in order to find the optimal policy. Then, we explain Generative Adversarial Networks(GAN) and the follow-up studies that improved GAN. Finally, we mention past key studies in the exploration problem and what our method contributes to this field.

2.1 Reinforcement Learning

In reinforcement learning, the learner, called the agent, interacts with an environment, which is the model of the problem. At a certain time step, the environment is represented by a state, which contains all the necessary information to fully describe the current environment. The environment is tasked with sending the observation of the current state to the agent at every time step, and the agent decides upon an action and performs it, and receives a numerical reward from the environment. The agent aims to learn how to select the best action to perform in order to get the highest possible reward. Figure 2.1 illustrates the cycle of these steps for a single time step. In the beginning, the agent does not receive any prior information; it does not know what each action does or which action to choose, what the observation means, and what reward it will receive. It needs to learn all the relevant information for maximizing the reward during the learning process by performing the actions and observing the results. In addition to the immediate results and reward, the performed action may also affect the future observation, thus the future rewards as well. As stated by Sutton and Barlo, what distinguishes reinforcement learning the most from other areas of artificial intelligence are these two characteristics: the trial-and-error manner of learning, and the actions in the present affecting the future rewards [6].

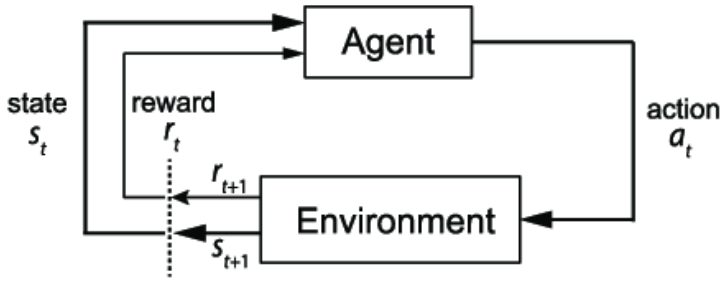

Figure 2.1 : An illustration of a single time step in reinforcement learning. After the observation of state s_t is received from the environment, the agent decides on an action a_t among the possible actions set and the environment sends the agent a reward r_t. The policy that maximizes the sum of rewards is the policy the agent aims to learn. Original image taken from [6].

Environments in reinforcement learning are modeled by Markov Decision Process (MDP) [22]. MDPs are a special type of Markov Chains. Similar to Markov Chains, MDPs carry the "Markov property", i.e. given the current state, the future states are not affected of the states and actions from the history. In other words, a state at the time step t, s_t, carries all the necessary information for the agent to make the optimal decision, and it uniquely describes the current state of the environment. Mathematically, this can be formulated as:

$$P[s_{t+1}|s_1, s_2, ..., s_t] = P[s_{t+1}|s_t] \tag{2.1}$$

Extending the Markov Chains, MDPs feature decision-making for the agent. The next state is not only dependent on the current state, but also on the action the agent selects, such that:

$$P[s_{t+1}|s_1, s_2, ..., s_t, a_t] = P[s_{t+1}|s_t, a_t] \tag{2.2}$$

where a_t is the performed action at time step t. This decision-making allows the agent to traverse through the optimal state sequence in order to maximize the reward gain. More compactly, MDP can be defined as a tuple of sets and functions, such that $\mathbb{M} = (\mathbb{S}, \mathbb{A}, \mathbb{R}, \mathbb{P}, \rho_0)$, where $\mathbb{M}$ denotes the MDP, $\mathbb{S}$ the set of possible states, which can be a finite or infinite set, $\mathbb{A}$ the set of actions, which also can be finite or infinite, $\mathbb{R}$ the reward function, $\mathbb{P}$ the state transition probability distribution, i.e. the probability distribution of the possible next states given the current state and the selected action,

and ρ_0 the distribution of the initial state. During the learning process, the agent learns an action policy $\pi(s)$. At each time step, the agent observes a state s_t, makes a decision about the optimal action to take according to its learned action policy $\pi(s_t)$ and selects an action a_t and receives a scalar reward r_t. According to the state transition determined by $\mathbb{P}$, the next state s_{t+1} is observed. As stated before, the agent aims to maximize the rewards, and to do so, it needs to find the optimal policy, which is the main goal of the reinforcement learning, such that

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi(s_t),\mathbb{P}} \left[\sum_t \gamma^t r_t \right] \tag{2.3}$$

where π^* is the optimal policy and $\gamma \in [0,1)$ is the discount factor. The aim of the discount factor is to determine the weight of the future rewards affecting our current decision making. If the discount factor is 0, the agent only takes the immediate reward into the consideration and will not learn to make a decision based on any future reward, while as the discount factor gets closer to 1, rewards in the more distant future get larger impacts on the decisions of the agent.

In reinforcement learning, a value of state s is defined as the expected total reward received using the policy of the agent from s. Mathematically, for any state, we can define it as:

$$V(s_t) = \mathbb{E}_{\pi(s_t),\mathbb{P}} \left[\sum_k \gamma^t r_{t+k} \right]. \tag{2.4}$$

where t denotes the discrete time intervals. Note that this equation can also be written as

$$V(s_t) = r_t + \gamma \mathbb{E}_{\pi(s_{t+1}),\mathbb{P}} \left[\sum_k \gamma^t r_{t+k} \right] \tag{2.5}$$

which is equivalent to

$$V(s_t) = r_t + \gamma V(s_{t+1}). \tag{2.6}$$

The importance of this derivation is that, in order to update the policy using Eq. 2.3, we need to collect the rewards until the end of one episode in the environment, while

with Eq. 2.6, if we store state values or can approximate them, we can update the policy at each time step, as soon as we observe a new reward.

2.2 Exploration in Reinforcement Learning

As the agent has no information about the environment initially, after a certain amount of training, the question of whether what it learned is enough to find the optimal solution arises. Since there is no definite metric of how much it needs to learn before it can find a generalized solution, the agent is also incentivized to explore the environment, heavily in the beginning but still occasionally even after it finds a solution, in case a more optimal solution exists in the environment. However, exploring may not yield any better solution as well, therefore a certain amount of time and computing power may be wasted in favor of exploring against exploiting the found solution. Moreover, as the agent learns by receiving a reinforcement through the numerical rewards, and in reinforcement learning, there exist problems that feature environments with very sparse reward or no reward settings, trying to find a solution that takes the agent to the goal by randomly exploring will most definitely be in vain. Because of these two reasons, i.e. unnecessary explorations and sparse reward settings, many studies continue to focus on efficient exploration, as this problem remains open as an active study area. The need for exploring efficiently has been acknowledged since the beginning of the 1990s [23]–[26]. However, with the emergence of deep learning, it became possible to solve much more complex problems with reinforcement learning, due to the usage of artificial neural networks as the function approximators making it possible to learn the mapping of state to actions in much higher dimensions. Due to this development, classic exploration approaches, such as ε-greedy [10], noise-based exploration [11], maximizing entropy [12], Boltzmann exploration [23], Thompson sampling [27] and so on, have not been as effective because of the dependence of randomness in these methods. Most of the recent studies tackled the idea of providing an extra reward, called "intrinsic" reward, to the agent. While the reward from the environment, which is also called "extrinsic" reward in this context, gives feedback to the agent related to the goal, the intrinsic reward incentivizes the agent to explore the environment by assigning high rewards

to the states that we'd like the agent to visit. Inspired by Weng [28], in this thesis, we categorize such works as prediction-based exploration, count-based exploration, random networks and memory-based exploration. **Prediction-based exploration** models try to learn the dynamics of the environment , and use this capability to predict the future or the outcome of certain events, and this prediction is used to assign intrinsic rewards. One of the most popular concepts in this area is "curiosity", where the model would lead the agent to novel states by assigning higher rewards to the states, the future of which it can not predict well. The idea of curiously acting has been explored earlier by Schmidhuber [29], and recent studies have adopted this trend to the modern problems. Stadie et al. used autoencoders [30] to encode the state space, and trained a dynamic model using the encoded states [13]. Improving the encoding idea, Pathak et al. suggested that the unnecessary details in the states, such as background elements that do not affect the optimal action, should also be cleared, and trained an inverse dynamic model to encode the states, which only captures the necessary information to make a decision [14].

Count-based exploration focuses on counting the number of times the agent observes a state, and assigning higher intrinsic rewards to the states that are visited less amount of times. As most modern problems feature a very high dimensional state space, recent studies have worked on approximating the number of visitations. Bellemare et al. benefitted from a density model to approximate how frequently states are visited, and proposed a novel model that outputs a "pseudo-count" for a given state [15]. Ostrovski et al. [16] and Zhao and Tresp [17] has utilized PixelCNN [31] and Gaussian Mixture Model, respectively, to be used as the density model in a similar fashion. Instead of approximating the frequency, Tang et al. used hashing to shrink the state-space to keep track of the number of times a state is visited [18].

In **random network approaches**, a new task, unrelated to the original one, is introduced to the problem. Depending on the model's performance on this task in the current state, an intrinsic reward is computed. In Directed Outreaching Reinforcement Action-Selection(DORA), Choshen et al. created a second MDP that is identical to the one at hand, except that the second MDP carries no reward at all. In this MDP, the value of any state is always assumed to be zero, because none of the states can lead the

agent to a reward [19]. Depending on how well the proposed model predicts the state value to be zero, it assigns an intrinsic reward to the current state. If it can not make a good prediction, i.e. it can not predict the value to be zero, the state is not well-known and should be visited more, therefore a high reward is assigned. In Random Network Distillation(RND) [4], a neural network that takes a state as input is initialized and the initial values are fixed. A second network, that also takes a state as input, is tasked with estimating the output of the fixed network. If the difference between the output of these two networks is small for a given state, that state is visited many times and well known to the agent, otherwise a high intrinsic reward is assigned to it. **Memory-based exploration** methods tend to fix the problems of the intrinsic reward models, such as forgetting the past experience, or repeating a novel experience in a short time span due to its high reward, by introducing a memory model. One of the most successful methods in this category is the Never Give Up(NGU), where the authors combine RND with a memory-based episodic reward module which keeps track of the embeddings of the visited state in the current episode, in order to discourage the agent to visit the same states in a single episode, while still using RND for long-term exploration [3].

There are also direct-exploration methods that focus on exploring as the main task instead of computing an intrinsic reward for the visited states. In Go-Explore, proposed by Ecoffet et al., the agent is tasked with only exploring before policy learning starts, until a solution is found [2]. The exploration is not entirely random, a memory of encoded states is maintained to sample from, and periodically, the agent goes back to a sampled state from the memory and continues the exploration. The sampling is probabilistic with an added heuristic that allocates higher probability to the states that may lead to a good exploration. Once exploration is complete and a solution is found, the solution is adopted by the agent through imitation learning. Another method called "Diverse Trajectory-conditioned Self Imitation Learning (DTSIL)" stores trajectories that leads to less frequently visited states and trains a policy using Self-Imitation Learning [32] to imitate these stored trajectories [5].

Our proposed method follows the trend of computing an intrinsic reward for novel states and feeding them to the agent. To do so, we utilized GANs to capture the distribution of the visited states and assign high rewards to unvisited, novel states.

By using GANs, we can train a model that works by only learning the distribution of the observed states, without learning anything about the dynamics of the environment. In addition, GANs are widely studied and show similarities to reinforcement learning. We believe integrating GANs into reinforcement learning can take this area of study further and promote the cooperation of GANs and reinforcement learning.

2.3 Artificial Neural Networks

Deep learning has been a rising trend in recent years and the artificial neural networks (ANN) have been the backbone of it. ANNs are models that are inspired by how the human brain and the biological neurons work. Even though deep learning is relatively new, the idea of ANNs is not; it's been discussed since the 1940s. McCulloch and Pitts proposed the first computational model that leads to the invention of ANNs [33] while Rosenblatt created the first ANN [34]. The popularity of ANNs has risen with the computational power increasing and the amount of data that can be accessed. Figure 2.2 shows a diagram of an ANN. Each of the circles represents a neuron, and a column of neurons forms a layer. Neurons in each layer have a connection to the neurons in the previous and the next layers, and each connection has a weight of its own. Starting from the input layer, the left-most one, values carried in the neurons in each layer is propagated to the next layer. The value of a neuron from the next layer is computed by multiplying the weights from the connection to itself with each neuron from the previous layer, and then summing them all together. The final value is then passed through an activation function, usually a non-linear one. In a vectorized form, propagation from one layer to another can be shown as:

$$X^{i+1} = f^i(W^i, X^i + bi) \tag{2.7}$$

where X^i is the values of the neurons in the i^{th} layer, W^i is the matrix composed of the weights of the connections from i^{th} layer to $i+1^{th}$ layer, bi is the extra bias term and f is the activation function.

Later studies started to call the type of layer mentioned above a "fully-connected" layer as different types of layers of ANN are developed. One of the most impactful

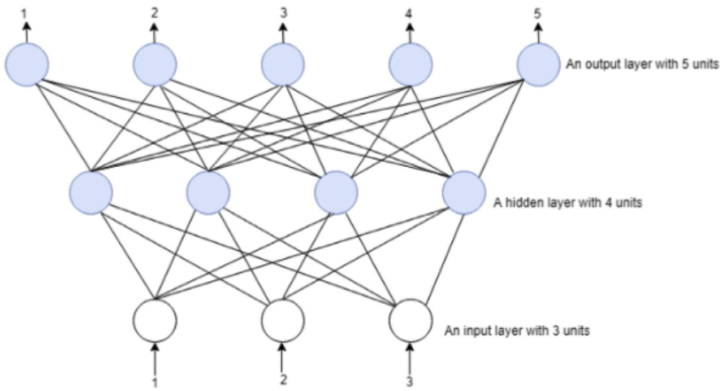

Figure 2.2 : A two-layered fully-connected artificial neural network. The image is taken from [7]

ones to the latest developments is the convolutional layer used in Convolutional Neural Networks(CNN). The convolutional layer has been preferred because of its ability to capture features from the image data, as it can capture the relations in an area better than the fully-connected layer. In the convolutional layer, the connections between neurons are replaced with filters, and the forward propagation from one layer to the next one is done through the convolution operation. Each filter in the layer is slid across the rows and columns, and the sum of the element-wise multiplication at each coordinate becomes the new value at the next layer. There can be multiple filters at the convolutional layers, and how many channels there will be in the output is determined by the number of filters. Filters are usually square shaped, and the size of it is depicted as kernel size, i.e a filter with a kernel size 3 is shaped $3x3$.

2.4 Generative Adversarial Networks

Proposed by Goodfellow et al., Generative Adversarial Networks(GAN) is a framework that consists of two artificial neural network models, namely the generator G and the discriminator D. In a typical GAN training, as depicted in Figure 2.3, these two networks compete against each other in order to train G to be able to generate synthetic data that is indistinguishable from real data. D is trained in order to detect whether a data is real or fake (generated by G), while G is trained with the aim of generating data that can fool D. In the work of Goodfellow et al., D is optimized to maximize the objective $\log D(\boldsymbol{x}) + \log(1 - D(G(\boldsymbol{z})))$ while G is optimized to minimize $\log(1 - D(G(\boldsymbol{z})))$, where $\boldsymbol{x}$ represents the real data, $\boldsymbol{z}$ represents a noise vector sampled

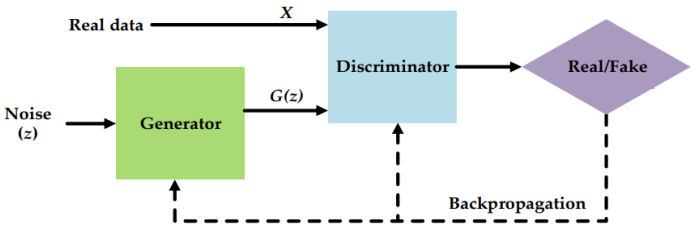

Figure 2.3 : A simple depiction of Generative Adversarial Networks. Generator aims to generate synthetic data that resembles the data at hand, while the goal of the discriminator is to discriminate these two types of data. Image taken from [8]

from a noise distribution p_z, $G(z)$ represents the synthetic data generated by G and $D(x)$ is the probability of x being real data, between 0 and 1. In an optimal training scenario, G is trained to learn the distribution of the real data and is capable of generating synthetic data indistinguishable from real data. GAN's optimization relies on the idea of a Nash equilibrium in a zero-sum game, where, in the end, ideally, D will have a %50 probability of detecting the fake image, and none of the networks overwhelmingly performs better than the other (even though Farnia and Ozdaglar discuss the idea that GANs may not have a Nash equilibrium in their work [35]).

The success of GANs brought a lot of new studies, improving and creating new ways of utilization, almost making GANs a sub-area of deep learning by itself. One of the very first studies is Deep Convolutional GAN, where the authors combine Convolutional Neural Networks(CNN) with GAN in order to enhance the performance of GAN on image datasets, while also using its capabilities for extracting features to use in various tasks [36]. Another work by Mirza and Osindero proposed Conditional GAN, in which they show that GAN can be conditioned on which features the generated data will have. Improvement to the original GAN loss has also been a focus of some works, one of the best performing one being Wasserstain GAN (WGAN), where the authors optimize GAN by minimizing an approximation of Earth Mover distance with the aim of fixing the problems of the original GAN loss [37]. Work on WGANs has been extended with improvements, such as WGAN-GP [38] and WGAN-DIV [39].

Schlegl et al. [20] utilized GANs in an anomaly detection task. Their work proposes to use GAN's ability to represent and learn the distribution of the data it is trained with, and detect the data that does not belong to the learned distribution. First, they

train a GAN with the normal, non-anomaly data, until the convergence. Given a query image, they hypothesize that if they could find the noise sample z that could generate the image that resembles the query image the most, the metric score of how different the query and the generated image are can be used to detect the anomalies. If the query image is an anomaly, it does not belong to the distribution of the normal data, therefore the difference will be high. To find z, they sample a random z and run an iterative optimization process to update to z to minimize the difference between the query and the generated image, and the discriminatory features of them. Schlegl et al. [21] proposed an improved version of this work, where, instead of running an optimization on noise sample, they train another network, called encoder, to map the query image to the z. Training of the encoder is done to minimize a similar loss to the one in the optimization of the noise z.

3. EXPLORATION GUIDED BY GAN-BASED INTRINSIC REWARD MODULE

In this chapter, inspired by the anomaly detection framework "f-AnoGAN" of Schlegl et al. [21], we present a new module that utilizes the GAN framework in order to lead the reinforcement learning agent into an efficient exploration, called **GAN-Based Intrinsic Reward Module (GIRM)**. To our knowledge, this is the second study that utilizes GAN in an exploration problem, the first one being the work of Hong et al. [1], in which, after training the GAN, they try to utilize the discriminator by feeding it a state input, and using the output to calculate a reward, the idea being the discriminator D will be able to distinguish visited states from novel ones. The core problem with this approach is that D is trained to distinguish the real data from the synthetic data generator G generates, however, D's behavior on the unseen instances of data can not be known beforehand, meaning that D may not be able to distinguish visited states from the novel ones. Moreover, when the GAN converges, D and G start to not beat each other, i.e. D starts to be uncertain about the probability of the input being real or fake, even though the data is from the original training dataset. In their work, Hong et al. tried to overcome this problem by not training G very optimally by stopping its training periodically, but to train a model trying to minimize an objective function while at the same time trying to stop the model from being "too good" is a troublesome task. Our method does not encounter these two mentioned problems because the GAN is trained to convergence, meaning that G is optimized to be able to generate real-looking synthetic data, and we use G's capability of learning the distribution of real data to distinguish novel states and assign high rewards to them. Our method, GIRM, uses the GAN in the way it is supposed to, and does not encounter any drawbacks due to its training objectives. Section 3.1 explains how GIRM does that in more detail.

3.1 Main Framework

In this section, we give details on the reinforcement learning framework with the addition of our proposed module, GIRM. GIRM incentivizes the reinforcement learning agent to explore the novel, unvisited states by providing an additional intrinsic reward r_t^i in addition to the extrinsic reward received from the environment, r_t^e. The aim is to compute the intrinsic reward so that in the unvisited states, the intrinsic rewards will be higher than the ones in the frequently visited, known states. With the addition of intrinsic reward, the reward at the time step t then becomes $r_t = r_t^e + r_t^i$. Since the agent will begin to receive higher rewards for the novel states, it will learn to traverse through those states, leading to exploring new parts of the environment it has not covered before. This is especially important for the environments that feature very sparse rewards, or no rewards at all. This is due to the fact that without additional reinforcement, the agent usually fails to reach the states that have rewards in it. When the agent has not been able to get any rewards at the beginning of its learning, the specific set of actions that will lead the agent to a goal state can only be selected randomly, which is near impossible for modern problems as this means perfect action selection for over thousands of steps. With the help of GIRM, even if the agent does not receive any extrinsic rewards, it begins to receive a reward at every time step through GIRM, which reinforces the agent to explore the environment, helping it find the goal state or states with extrinsic rewards. Figure 3.1 illustrates the proposed GIRM-enhanced reinforcement learning framework.

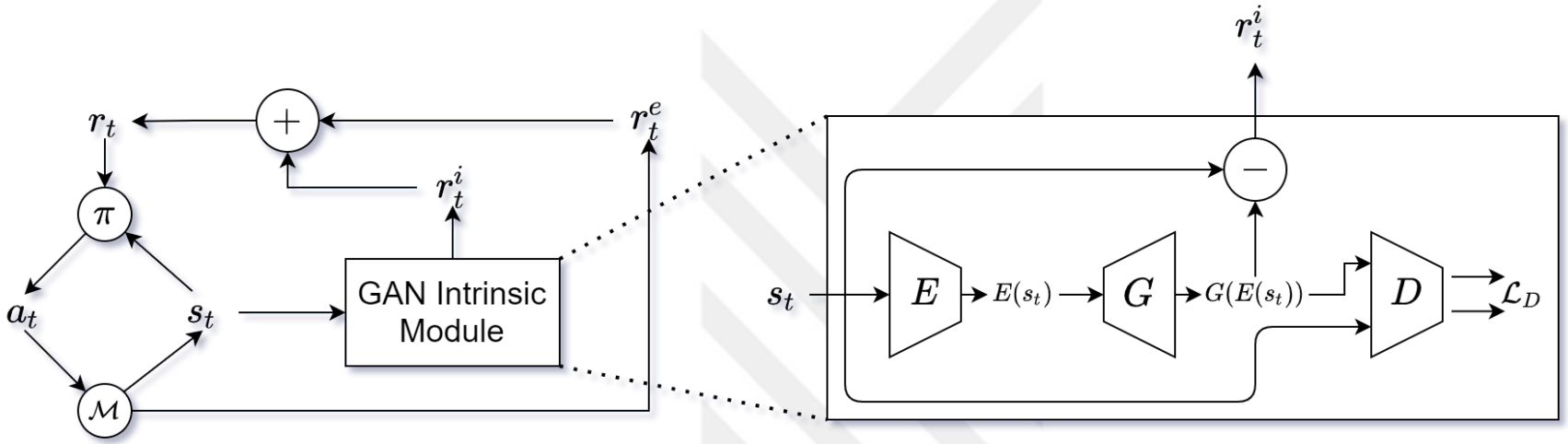

Figure 3.1 : A general reinforcement learning framework with our proposed GAN-based Intrinsic Reward Module (GIRM). At each time step t, agent observes a state s_t from the environment MDP $\mathscr{M}$, chooses an action depending on its policy π, and receives an extrinsic reward r_t^e from the environment and an intrinsic reward r_t^i from GIRM. In GIRM, the encoder E maps s_t to the input noise space of the generator G and outputs $E(s_t)$. G generates a synthetic state $G(E(s_t))$ and the error measured between s_t and $G(E(s_t))$ is then sent to the agent as an intrinsic reward. Operation $-$ in the GIRM module is not necessarily subtraction, but could be any function that computes the residual between s_t and $G(E(s_t))$. $\mathscr{L}_D$ is used for training the Encoder.

The first step in the training of GIRM is to learn the distribution of the visited states. To accomplish this task, we create a training dataset from the observations of visited states, which are mostly images, and train a GAN using this dataset. In this work, we used the improved WGAN architecture [38], due to its better performance and the empirical results from Schlegl et al. [21] showing that WGAN is better suited for the anomaly detection task. As the generator G of the GAN is trained to minimize the divergence between its distribution and the distribution of the training data, G is optimized to learn the distribution of the observations of the visited states. In other words, we train G to learn the mapping $\boldsymbol{z} \rightarrow G(\boldsymbol{z})$, and in this framework, $G(\boldsymbol{z})$ being a synthetic data that resembles the visited states.

The second step consists of training an encoder network E to learn to mapping of the state space to the input noise distribution of G, i.e. $s_t \rightarrow \boldsymbol{z}$ where s_t is the state at time step t. This mapping is required to find the noise sample that generates the synthetic data $G(\boldsymbol{z})$ in the learned distribution of G as close to the input state s_t as possible. Once this mapping is learned, it is possible to recreate the state input s_t by propagating it through E and G networks, respectively. In the training, E is optimized to minimize a combination of two losses. The first loss is the MSE loss to minimize the difference between the input s_t and the recreated sample $G(\boldsymbol{z}) = G(E(s_t))$:

$$\min_{E} \mathscr{L}(s_t) = \frac{1}{n} \|s_t - G(E(s_t))\|^2 \tag{3.1}$$

where n is the number of dimensions in the state space. As a lot of problems now feature images as the state data, and in this work, we use such environments, n will represent the number of pixels. The MSE loss is the main objective in order to learn the mapping, however, Schlegl et al. [20] and Schlegl et al. [21] stated that a secondary discrimination loss to enforce the recreated image to lie in the learned distribution of G and to have similar discriminatory features of the real data improved the training of E. Moreover, discriminator D is capable of learning the feature representations of the data in its intermediate layers. Thus, following their idea, we also introduce a discrimination loss that minimizes the differences between feature encodings from the discriminator, and define this loss as $\mathscr{L}_D(s_t) = \frac{\lambda}{n_d} \|f(s_t) - f(G(E(s_t)))\|^2$ where f is the output of an intermediate layer in D, the n_d represents the number of dimension

in the output of this intermediate layer and λ is a hyperparameter to determine the weight of the discriminator loss. Combining both losses, we finalize the loss function for optimizing E as:

$$\min_E \mathscr{L}(s_t) + \mathscr{L}_D(s_t) = \frac{1}{n}\|s_t - G(E(s_t))\|^2 + \frac{\lambda}{n_d}\|f(s_t) - f(G(E(s_t)))\|^2 \tag{3.2}$$

After the training of E is complete, the framework has the tools to compute intrinsic rewards. As illustrated in Figure 3.1, at the time step t, the intrinsic reward for the state agent encounters is calculated as the MSE between s_t and $G(E(s_t)$:

$$r_t^i = \frac{1}{n}\|s_t - G(E(s_t))\|^2, \tag{3.3}$$

Since the training data of GAN consists of the data from the visited states, and the pipeline of E and G is only able to create the data from the training set, the difference, and therefore r_t^i will be low for the visited states. However, a novel state will not be in the distribution that the G learns, and no matter the mapping of E, the recreated image will not be similar if the input state is a novel one. In such states, as the difference between the $G(E(s_t))$ and the input s_t will be significant the intrinsic reward r_t^i will be computed as a higher value. With this intrinsic reward, the agent is incentivized to explore the novel states as the reward it receives will be higher.

3.2 Standardization of the Intrinsic Reward

The scale of the reward function is one of the factors that impact the performance of the model. Especially in deep reinforcement learning, changes in the scale may require different hyperparameter setups in order to converge to a good solution and our method so far does not provide any mechanism for the scale of the intrinsic reward. One of the possible solutions could be to multiply r_t^i with a constant, however with this approach, as the learning continues and new states are visited, the learned distribution of G will change. Therefore, instead of introducing a new scalar coefficient, we calculate the

exponentially weighted moving average(EMA) and variance(EMV) of the rewards as:

$$EMA = \begin{cases} r_1^i, & \text{if } t = 1 \\ \alpha * r_t^i + (1-\alpha) * EMA, & \text{otherwise} \end{cases} \tag{3.4}$$

$$EMV = \begin{cases} 0, & \text{if } t = 1 \\ \alpha * (r_t^i - EMA)^2 + \\ \quad (1-\alpha) * EMV, & \text{otherwise} \end{cases} \tag{3.5}$$

and then, standardize the intrinsic reward:

$$r_t^i = \frac{r_t^i - EMA}{\sqrt{EMV}}. \tag{3.6}$$

In this study, α is set to 0.01, however this value could change depending on the needs of the environment. With the standardization, we ensure that the scale of the intrinsic rewards calculated will not change throughout the training, leading to more stable learning in terms of the scale of rewards. In Eq. 3.6, the intrinsic reward is standardized to have a mean of 0 and a variance of 1, and in this study we used these values, however the equation can also be modified easily to have a different mean and variance, being flexible to the needs of the environment. After standardizing the intrinsic reward, the total reward for the agent is now calculated as $r_t = r_t^e + r_t^i$, and the optimization problem in Eq. 2.3 becomes:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi(s_t),\mathscr{P}} \left[\sum_t \gamma^t (r_t^e + r_t^i) \right]. \tag{3.7}$$

3.3 Periodic Training of GIRM

As the policy learning continues, the distribution of the visited states keeps changing. Moreover, with the exploration in effect, novel states will be observed and added to the distribution, which will change the distribution drastically. Due to this fact, GIRM needs to be updated during the training, as once novel states will be frequently visited states after a while, and GIRM should not keep giving high rewards to such states. To this end, GIRM is trained periodically during the policy learning. To do so, we use a state memory $M = s_0, s_1, ..., s_N$ with a predefined size of N that starts clear and stores the states as they are observed by the agent during the policy learning. Once M reaches full capacity, policy learning stops temporarily, and GIRM is trained with

the data in M using stochastic gradient descent, a popular optimization technique in deep learning [40] to train the GAN and minimize the objective function in Eq. 3.2 for training the encoder. After the training of the GIRM is completed, all the content stored in M is cleared, and the policy learning and the process of storing the observed states to M restarts. Every time M reaches full capacity, the periodic training of GIRM restarts while policy learning waits for the training of the GIRM. At each training cycle, GIRM is trained for a fixed number of epochs with the contents of M, before the contents are cleared. At each cycle, since M gathers new data, the distribution of the states learned by the GIRM keeps updating and converging to the actual distribution of the visited states.

Since at the beginning of policy learning, M does not have any content and the GIRM is not trained, the GIRM consists of only random networks, therefore the reinforcement received from it will be random. Therefore, until the first training cycle of the GIRM is completed, the agent is only trained using the extrinsic reward to avoid the catastrophic random effects to the learning occurring. Due to this, we use random exploration techniques before GIRM takes effect. In this work, we substituted GIRM for the first cycle with entropy maximization, which tries to maximize the entropy of the actions selected by the agent [12]. Moreover, the first time M reaches full capacity, GIRM is trained for a larger number of epochs to reach the convergence point. Afterward, GIRM is fine-tuned for a smaller number of epochs with new data stored in M.

4. EXPERIMENTS AND RESULTS

This chapter presents the experimental setup divided into the subsections for baseline reinforcement learning algorithm, the network architectures and the environments used for the experiments. The experimental setup is followed by the results of the experiments and their comparison with the results from other studies. Finally, the discussion of the results is presented.

4.1 Experimental Setup

4.1.1 The Baseline Algorithm

Since our module GIRM modifies only the reward signal, it is designed to be used with any reinforcement learning algorithm. For this study, we chose Advantage Actor-Critic (A2C) as our baseline algorithm to evaluate GIRM in the experiments, synchronous version of Asynchronous Advantage Actor-Critic (A3C) [12]. A2C belongs to the actor-critic algorithms, which feature two models: the critic estimates the value of a state while the actor is responsible for learning a policy that maximizes the collected reward using the value approximation of the critic. The actor aims to update its policy to direct the agent to the states with the highest value at each time step. In A2C, the actor is trained to learn a policy to maximize the "advantage" of an action instead of the value of the next state. The advantage is defined as $A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, which calculates how much better the next state will be if the agent performs a_t. Another property of A2C is that it runs multiple parallel workers sharing one global actor network, each of them giving out a transition simultaneously at each time step. In our experiments, we used 16 workers. We chose A2C for this study because its performance competes with the state-of-the-art results and is easy to implement, making it a good choice for benchmarking for the GIRM. To evaluate GIRM, we compare the performance of vanilla A2C with A2C combined with GIRM module, in addition to comparing both results with other studies. Note that vanilla A2C uses

entropy maximization as an exploration method, which means this study compares the performance of GIRM to one of the common exploration strategies as well.

4.1.2 The Network Architectures

In this work, the environments used for the experiments return the observations of the states as RGB images, which we convert to grayscale image and resize to $84x84$. Instead of two different neural networks for actor and critic, we use a single network with two different heads. The shared network of actor and critic acts as a feature extraction network and is composed of 3 convolutional layers and then a fully connected layer. The first convolutional layer has 32 filters of $8x8$ kernel size with stride 4, the second convolutional layer has 64 filters of $4x4$ kernel size with stride 2, and the final convolutional layer has 64 filters of kernel size $3x3$ with stride 1. The output of the final convolutional layer is then sent to a fully connected layer with 512 hidden units. The shared network's output is used as the input for two different heads, each of them consisting of a single fully connected layer. The head for the actor is for the action selection policy, while the head for the critic is for estimating the value of the input state.

As explained in Chapter 3, the GIRM consists of a generator G, discriminator D and an encoder E, all of which are modeled by convolutional neural networks. In this work, the size of the input noise sample for G is 128, which is created with 3 dimensions such that its size is $1x1x128$, which can be depicted as 128 activation maps with size $1x1$, like a convolutional layer input/output. G consists of 5 transposed layers; the first 4 layers consists of 64 filters with a kernel size of $4x4$, and the final layer is composed of a single filter with a $4x4$ kernel. The output of this network is an $84x84$ grayscale image. D consists of 6 convolutional layers; the first 5 layers consist of 64 filters and the final layer consists of a single filter. The kernel sizes of D are 4, 5, 5, 3, 1 and 5, respectively. The intermediate feature representation for the training of E, as explained in Section 3.1, is extracted using the first 5 layers of D. Finally, the network of E is formed with 5 convolutional layers, the first 4 consisting of 54 filters and the final one consisting of 128 filters, which is the size of the input noise of G, therefore the output of the E network is a $1x1$ mapping with 128 channels. Kernel sizes for E are

4, 5, 5, 3 and 5, respectively.All three networks use Batch Normalization [41] at the end of each layer except the final ones. Hidden layers in D and E are activated with Leaky ReLU with a negative slope of 0.2, and hidden layers of G use ReLU as the activation function, except the final layer which uses tanh as its activation function. For the optimization of all the networks, including the A2C networks, we used ADAM optimizer [42]. For training the actor-critic network, the learning rate is $2.5 * 10^{-4}$ and for the networks in GIRM module, the learning rate is 10^{-4}. $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for all the networks.

4.1.3 Environments

To evaluate the performance of the GIRM, we set up two environments. One of these environments is modified to have no reward signal, other than reaching the goal. In this environment, we compare the GIRM's performance with the baseline method. The second environment has sparse reward signals, and due to this, is known for being very hard to solve in the reinforcement learning field and many studies tried to solve this environment. We compare GIRM's performance in this environment with the baseline method and also with other studies.

4.1.3.1 Super Mario Bros

The first of the environments GIRM is evaluated on is Super Mario Bros. [43]. Super Mario Bros is a platform game released in 1983 and the environment of it has been used with reinforcement learning algorithms. An example visual from the game is given in Figure 4.1. The game features a character controlled by the player, and through the controls, it's possible to run and jump in the game. There are different types of enemies that can kill the character when contacted, or pits that when the character drops in, the character dies. There are also platforms where the character can jump on, and certain power-ups or coins the character can collect. The player is given a total of three lives, and the game restarts from the beginning when the player runs out of lives. In this study, we are only interested in the first level of the game, the layout of which is given in Figure 4.2. The player can only observe a part of the environment, centered around the controlled character, similar to the in Figure 4.1. The partial observation at time

Figure 4.1 : An example visual from Super Mario Bros. The game camera is centered a bit right to the character. There are blocks and green pipes as the platforms that can be jumped onto, mushrooms that gives power-ups, and mushroom-shaped enemies that can be killed by jumping onto.

step t is the state returned from the environment at t, which is an RGB image. The initial position of the character is at the far left of the level, while the goal point is at the far right side, next to the castle as seen in Figure 4.2. To create a no reward setup, we disabled all the extrinsic rewards in this level, except the one given for the goal state. In such an environment, an agent with an exploration strategy that depends on random action selection will fail to reach the goal state, as the character needs to go right constantly while avoiding all the obstacles and enemies in a totally random manner. Therefore, a smarter way of exploration is needed.

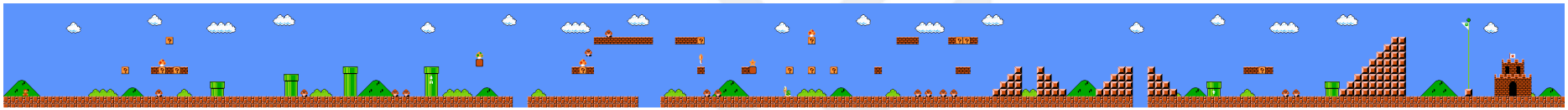

Figure 4.2 : A full view of the first level in the game Super Mario Bros. Player starts at the left side of the level and only perceives the surroundings of the character it is controlling, while the camera is always centered at the character in the horizontal-axis. To complete the level, the player is tasked with navigating the character to the castle at the far right of the level. With no reward given, a reinforcement learning agent only depends on exploration to find the end of the level.

4.1.3.2 Montezuma's Revenge

The second environment is Montezuma's Revenge, a platform game known for its difficulty that is released in 1984. Recently, it has been a benchmark environment for reinforcement learning algorithms as it is notoriously hard to solve due to the sparse reward signals and the complexity of the required sequence of actions to receive those rewards. In the game, the player is able to move right or left, jump, climb up or down stairs or ropes. There are 99 total rooms in the game, each one of them having its own obstacles and enemies. In the game, the only reward received is for collecting certain items or killing the enemies, which can only be done after collecting a specific item. Any entity associated with the reward contributes to the game score equally, so in terms of training an agent, the total reward gathered is equal to the game score the agent achieved. Figure 4.3 shows the first room the player starts the game in. In this room, the only reward the agent can receive is the key, so it needs to navigate to the platform on the left in order to collect the key. However, in Montezuma's Revenge, falling from heights results in death, so the agent needs to find the specific order of actions to reach the key. The agent needs to, in order, jump to the middle platform, climb down the ladders, jump to the rope, jump to the platform on the right, climb down the stairs, run left while jumping over the enemy, climb up to the platform and finally, jump to collect the key. Through this sequence of actions, there are no rewards to collect so the agent needs to figure out how to explore efficiently in this room, and once the agent collects the key, it still needs to figure out what to do next, which is going to one of the doors at the upper platforms, and the doors do not resemble the key and by itself a completely different task. In this scenario, the agent might start exploiting the only reward it gets, getting stuck in the very first room of the game.

A different aspect of Montezuma's Revenge compared to Super Mario Bros. is that the agent observes the whole room, and the observed frame does not move with the movement of the character. Unless the player moves to a different room, the background and the layout the agent observes is static, which means most of the observed image will stay the same between any two states in the same room. This

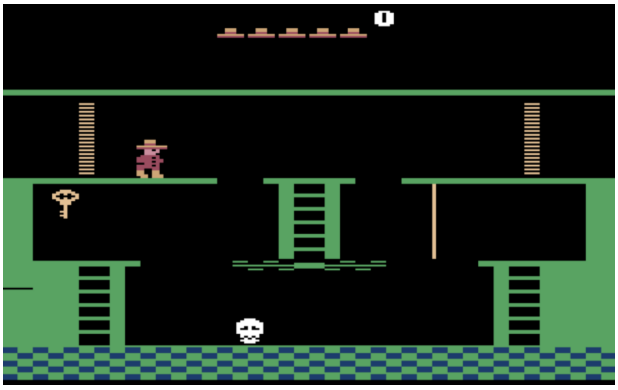

Figure 4.3 : First room in the game Montezuma's Revenge. The initial position of the character controlled in the game is the first room and the player needs to collect the key first, and then go to one of the doors to escape from the room. Only rewards are received when a key is grabbed or a door is unlocked. The player needs to complete a complex set of actions to reach the key, as it needs to climb down two stairs, jump to a rope, avoid an enemy, and then climb up a stair while avoiding death. The camera is static, the player has the full view of the room and it only changes when the player goes to another room.

means that until a new room is observed, the generated image from G will have the same background as the query state visual if the GIRM is trained ideally, which will lead to small, insignificant differences for r_t^i in Eq. 3.3. This problem is an example of why the standardization is needed. With standardization in Eq. 3.6, the small differences between visuals will have significant importance when computing the intrinsic reward r_t^i.

4.2 Results and Discussion

4.2.1 Results and Discussion of Experiments in Super Mario Bros.

We compare the performance of the GIRM with the action entropy maximization approach [12], which is the standard exploration approach of A2C in this environment. Table 4.1 presents the results of both approaches by the distance the agents covered in the first level of the game and the percentage of the visited states that are farther than 500^{th}, 1000^{th} and 1500^{th} units in the horizontal axis, in increasing order from left to right in the level (Note that end of the level is at the 3150^{th} unit). A2C denotes the vanilla approach with the random exploration, while A2C+GIRM denotes our approach, where the GIRM replaces the standard exploration method. As the results

show, when A2C is combined with the GIRM, it is able to reach the end of the level while being trained with 8 million frames. On the other hand, A2C without the GIRM was able to go near halfway point of the level, to the 1673^{th} point, while being trained with 32 million frames, four times as many frames as the GIRM used. Furthermore, of the visited states of vanilla A2C, only 0.1% corresponds to the states that are further than 1500^{th} point, which means that the agent can not reach the halfway of the level consistently.

As shown in Figure 4.2, since the agent starts the game at the far left side of the level and all the unexplored parts lie on the right side, the optimal strategy to explore is to always try to go further right of the level. In an environment where there is no extrinsic reward, the only thing that is interesting to the agent is observing new environmental elements, and vertical movements in Super Mario Bros. do not provide observing any novel parts and are only useful to overcome enemies and obstacles. As long as the path is clear, the agent can maximize the effectiveness of the exploration by going right. This strategy is achievable through the usage of the GIRM, as the unobserved states on the right side of the level will be assigned higher intrinsic rewards and the agent will eventually learn to go right. As shown in Table 4.1, 60.3% of the visited states are before the 500^{th} point of the level while 4.1% of the visited states are after the 1500^{th} point. It might seem that this result is not a desirable result because a small percentage of the visited states are at the second half of the level. However, the agent is still training and has not learned to explore efficiently yet at the beginning, therefore it has an ever-changing policy and observed most part of the beginning of the level. As both the GIRM and the agent keeps learning, the agent eventually learns an efficient exploration strategy, which is to go right as much as possible. Once this strategy is learned, the agent is capable of finding the end of the level without roaming around in the second half of the level. This shows that the reinforcement learning agent, when coupled with the GIRM, is capable of learning the exploration strategy the environment requires. This trait is an important one for the GIRM to have, as there is not one general exploration strategy that works in all environments, but each environment setting requires different exploration strategies, and the GIRM is capable of learning such strategies.

Table 4.1 : Comparison of agents in Super Mario Bros. in terms of furthest point they explored and percentage of the states that are further from certain point out of all of the states they visited in horizontal axis.

Method	# of Frames	Furthest Visited	visited > 500	visited > 1000	visited > 1500
A2C	32M	1673	41.3%	2.3%	0.1%
A2C+GIRM	8M	3150 (End of level)	60.3%	21.5%	4.1%

4.3 Results and Discussion of Experiments in Montezuma's Revenge

Being a notoriously hard-to-solve environment due to its sparse reward setting, efficient exploration is especially important in Montezuma's Revenge as the agent needs to gather the feedback from the environment to learn how to solve it. Figure 4.4 compares the mean scores of multiple runs gathered in the game with the agent trained with vanilla A2C and A2C+GIRM. As the results show, when GIRM is introduced, the agent starts to get higher scores than the vanilla method and is capable of reaching reward states, while without the GIRM, the agent could not get to any of the reward states, scoring a total of 0 most of the times. The vanilla agent was capable of grabbing the key in the room a few times, which is the first reward the agent can get, and in those cases could not get out of the room to receive further rewards. Furthermore, it wasn't able to reproduce the same action sequence to gather that reward. On the other hand, the agent trained with the GIRM has shown that it is capable of grabbing the key and getting out of the room, further exploring the environment. The first 500 score of reward is associated with getting out of the room, and the scores show that the agent is capable of receiving further rewards from the other rooms as well. However, even though the agent shows its capability of exploring, the fact that the algorithm converges early to a solution should be pointed out as the agent was not able to break out of the 3600 reward limit. Our observation of the agent's behavior showed that the agent discovered an enemy that gives 3000 reward for killing it and started exploiting this behavior for a significantly higher reward than anything else it managed to find up until that point. This indicates that, even though the GIRM introduces intrinsic rewards for exploring, the agent may still learn to exploit high extrinsic rewards when the difference between these extrinsic rewards and the intrinsic rewards is significantly high.

Table 4.2 shows the results of our method and other exploration studies in the field. The results further show that the GIRM is capable of incentivizing exploration and matching the performance of some of the other works. However, it is observed that the performance of the state-of-the-art was not beaten when the game score is taken

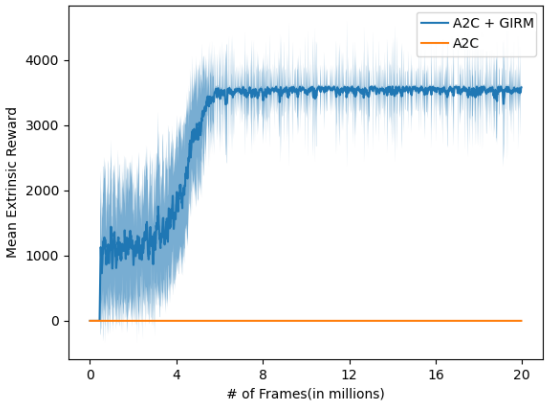

Figure 4.4 : Mean scores during the training of A2C and A2C + GIRM in Montezuma's Revenge..

Table 4.2 : Comparison of our results in Montezuma's Revenge against DQN-GAEX [1], Go-explore [2], NGU [3], RND [4] and DTSIL [5]. Scores are reported by the authors of the methods, except the baseline result from A2C and our method, GIRM. Bold indicates the best result.

Method	# of Frames	Mean Score
A2C	20M	0
A2C+GIRM	20M	3,594
DQN-GAEX	200M	420
Go-explore	12B	**43,791**
NGU	35B	10,400
RND	16B	10,070
DTSIL	3.2B	22,616

into consideration. It should be noted that the best performing methods have trained their models with billions of frames (between 3.2 billion and 35 billion frames), most probably due to their hardware capabilities being much larger than ours.

We also identify another factor that affects the performance of the GIRM in Montezuma's Revenge, which is the static background observations in the rooms. When the agent spends time in a room, it will observe the same background, which will fill the memory buffer of the GIRM with the same background, resulting in the trained generator G learning to generate the same background. In this case, the difference between the query state observation and the re-generated state image will be very small even if the position of the character is in entirely different places. This problem is addressed by introducing the standardization through EMA and EMV in Section

3.2. If the differences are consistently small, the standardization ensures that even the small differences impact the intrinsic reward significantly. However, when the agent observes a new room, two problems arise in order: 1) instant layout change due to the new room results in a very high numeric intrinsic reward, due to standardization, and 2) as the EMA and EMV are adjusting to new high rewards, small differences begin to not have any significant effect in the intrinsic reward. These two problems do not happen at the same time, but follow one another, and as both the GIRM goes through a new training stage and EMA and EMV adjust, both problems vanish. However, they create noise intrinsic reward during the time they are present. Nevertheless, as shown by the results in both Super Mario Bros. and Montezuma's Revenge, our method outperforms the commonly used random exploration approaches heavily, however, we also state that the aforementioned problems affect the performance of the GIRM in the negative direction.

4.3.1 Sample Efficiency of GIRM

As described in Section 3.3, the training of the GIRM occurs when the state memory M gets to full capacity. Moreover, in the first training phase, because the GIRM is not trained yet and consists of random networks, the capacity of M is larger and the GIRM is trained for a larger number of epochs in order to reach near convergence levels. This results in the GIRM having well-optimized networks to the visited states even after the first training phase and in turn, the GIRM becomes capable of computing intrinsic reward in the desired fashion such that the intrinsic rewards are high for the unvisited, novel states and low for the frequently visited states. Since the GIRM starts feeding a good reinforcement signal after the first training phase, it can be said that the amount of data it initially needs to incentivize exploration is the capacity of the M. This indicates the sample efficiency of the GIRM as further data is only needed to fine-tune the networks to learn the newly visited states. As shown in Table 4.1 and 4.2, A2C coupled with the GIRM is trained with only 8 million and 20 million frames for each respective environment, while the unsuccessful exploration of vanilla A2C took 32 million frames in Super Mario Bros. and the other, best performing studies for the exploration problem uses billions of frames to achieve the scores they got. We

conclude that our method, the GIRM, shows success in providing a good reward signal to incentivize exploration while being sample-efficient.

5. CONCLUSIONS AND FUTURE WORK

Reinforcement learning is known to be sample-inefficient due to its dependency on a good exploration phase as the agent might collapse to a sub-optimal solution without it and there is a need for experiencing the same result multiple times to learn, thus the agent needs to find the solution multiple times via the help of exploration. Common exploration approaches produce satisfying results if the rewards are dense in the environment, however, they begin to fail if the rewards are sparse or none at all. In such environments, a better, smarter exploration strategy is needed. To this end, this thesis presents a method that utilizes Generative Adversarial Networks(GAN) in the proposed "GAN-based Intrinsic Reward Module(GIRM)". The aim of the GIRM is to compute intrinsic rewards for each visited state. The GIRM is trained so that the intrinsic rewards are high for unvisited states, and low for frequently visited states, and through these rewards, the agent will be incentivized to explore the unvisited parts of the environment. The training of the GIRM is executed so that it fits the distribution of the visited states and can detect the out-of-distribution states, which are unobserved, and assign high rewards to those. The deployed memory mechanism in the GIRM helps with a stable training throughout the policy learning and the constant adaptation to newly visited states. Standardization of the intrinsic rewards ensures that the rewards will stay in a certain range, not disrupting the training of the policy learning agent. The GIRM is designed so that it can be used with any reinforcement learning algorithm. In this thesis, A2C is used as the baseline algorithm and to compare the performance of the GIRM with other strategies.

Performance of the GIRM is evaluated in a sparse reward and a no-reward environment. For the no-reward environment, the game Super Mario Bros is used. We showed that the GIRM is capable of providing the necessary intrinsic reward to incentivize the agent to explore the environment and find the goal state. Comparing with the baseline approach, we identify the weakness of the common exploration

approaches and how the GIRM overcomes that problem, outperforming the agent trained without it. Through the results, we also deduce that the GIRM is capable of learning the exploration strategy required in the environment it's in. For the sparse reward setting, Montezuma's Revenge, a benchmark environment in reinforcement learning studies due to its notorious difficulty, is used. We show that the baseline approach can not receive any reward and is stuck in the beginning in the room, while the agent trained with the GIRM learns to traverse multiple rooms and find the states that return positive rewards. Reported results show that the GIRM achieves a mean score of 3954. While this score beats the score of the baseline approach, A2C, which is 0, the GIRM could not outperform the state-of-the-art studies, because the GIRM could not avoid the agent exploiting large rewards in the environment, getting stuck to a certain pattern of actions in the environment. Nevertheless, the comparison with other studies shows the sample efficiency of the GIRM, using nearly one-hundredth amount of data the other studies used.

Standardization of the intrinsic rewards converts the small differences into significant ones, however, in Montezuma's Revenge, we identified a weakness of this method. As the GIRM manages to explore new rooms in the game, the difference between the learned distribution and the observed state gets multiple factors higher, and as the moving average and variances begin to adapt to this change, small differences begin to be insignificant, and the novel states in the same rooms are not rewarded correctly, but rather treated as any other state in the same room by the GIRM. A potential solution could be to discard the very high or low intrinsic rewards while calculating moving average and variance, treating them as an outlier. Another method could be to shift the distribution represented by the moving average and variance according to the computed intrinsic rewards. In the future, we aim to investigate this problem thoroughly.

The GIRM only learns the distribution of the state, however, GANs can also be integrated into a method to learn the dynamics of the environment. Dynamics depend on a set of rules in the environment, and they do not drastically change throughout, meaning if a method learns the dynamics once, it can use that information to evaluate the novelty from one end to another in the environment, without a need to adjusting. This can introduce a better generalization property to the model. As

this thesis demonstrates one way of incorporating GANS into reinforcement learning, it encourages future studies on how GANs can be used in the reinforcement learning, either as direct improvements to the GIRM framework or as in other interesting ways to extend utility of generative methods in exploration strategies of reinforcement learning.

REFERENCES

[1] **Hong, W.**, **Zhu, M.**, **Liu, M.**, **Zhang, W.**, **Zhou, M.**, **Yu, Y. and Sun, P.** (2019). Generative Adversarial Exploration for Reinforcement Learning, *Proceedings of the First International Conference on Distributed Artificial Intelligence*, DAI '19, Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3356464.3357706.

[2] **Ecoffet, A.**, **Huizinga, J.**, **Lehman, J.**, **Stanley, K.O. and Clune, J.** (2021). First return, then explore, *Nature*, *590*(7847), 580–586, https://doi.org/10.1038/s41586-020-03157-9.

[3] **Badia, A.P.**, **Sprechmann, P.**, **Vitvitskyi, A.**, **Guo, Z.D.**, **Piot, B.**, **Kapturowski, S.**, **Tieleman, O.**, **Arjovsky, M.**, **Pritzel, A.**, **Bolt, A. and Blundell, C.** (2020). Never Give Up: Learning Directed Exploration Strategies, *CoRR*, *abs/2002.06038*, https://arxiv.org/abs/2002.06038, 2002.06038.

[4] **Burda, Y.**, **Edwards, H.**, **Storkey, A. and Klimov, O.** (2019). Exploration by random network distillation, *7th International Conference on Learning Representations (ICLR 2019)*, pp.1–17, https://iclr.cc/, seventh International Conference on Learning Representations, ICLR 2019 ; Conference date: 06-05-2019 Through 09-05-2019.

[5] **Guo, Y.**, **Choi, J.**, **Moczulski, M.**, **Bengio, S.**, **Norouzi, M. and Lee, H.** (2019). Efficient Exploration with Self-Imitation Learning via Trajectory-Conditioned Policy, *ArXiv*, *abs/1907.10247*.

[6] **Sutton, R.S. and Barto, A.G.** (2018). *Reinforcement Learning: An Introduction*, The MIT Press, second edition, http://incompleteideas.net/book/the-book-2nd.html.

[7] **Abiodun, O.I.**, **Jantan, A.**, **Omolara, A.E.**, **Dada, K.V.**, **Mohamed, N.A. and Arshad, H.** (2018). State-of-the-art in artificial neural network applications: A survey, *Heliyon*, *4*(11), e00938, https://www.sciencedirect.com/science/article/pii/S2405844018332067.

[8] **Feng, J.**, **Feng, X.**, **Chen, J.**, **Cao, X.**, **Zhang, X.**, **Jiao, L. and Yu, T.** (2020). Generative Adversarial Networks Based on Collaborative Learning and Attention Mechanism for Hyperspectral Image Classification, *Remote Sensing*, *12*, 1149.

[9] **Goodfellow, I.**, **Pouget-Abadie, J.**, **Mirza, M.**, **Xu, B.**, **Warde-Farley, D.**, **Ozair, S.**, **Courville, A. and Bengio, Y.** (2014). Generative Adversarial Nets, ***Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K.Q. Weinberger****, editors, Advances in Neural Information Processing Systems*, volume 27, Curran Associates, Inc., https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[10] **Mnih, V.**, **Kavukcuoglu, K.**, **Silver, D.**, **Graves, A.**, **Antonoglou, I.**, **Wierstra, D. and Riedmiller, M.A.** (2013). Playing Atari with Deep Reinforcement Learning, *CoRR*, *abs/1312.5602*, http://arxiv.org/abs/1312.5602, 1312.5602.

[11] **Lillicrap, T.P.**, **Hunt, J.J.**, **Pritzel, A.**, **Heess, N.**, **Erez, T.**, **Tassa, Y.**, **Silver, D. and Wierstra, D.** (2016). Continuous control with deep reinforcement learning., ***Y. Bengio and Y. LeCun****, editors, ICLR*, http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

[12] **Mnih, V.**, **Badia, A.P.**, **Mirza, M.**, **Graves, A.**, **Lillicrap, T.**, **Harley, T.**, **Silver, D. and Kavukcuoglu, K.** (2016). Asynchronous Methods for Deep Reinforcement Learning, ***M.F. Balcan and K.Q. Weinberger****, editors, Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, PMLR, New York, New York, USA, pp.1928–1937, https://proceedings.mlr.press/v48/mniha16.html.

[13] **Stadie, B.C.**, **Levine, S. and Abbeel, P.** (2015). Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models, *CoRR*, *abs/1507.00814*, http://arxiv.org/abs/1507.00814, 1507.00814.

[14] **Pathak, D.**, **Agrawal, P.**, **Efros, A.A. and Darrell, T.** (2017). Curiosity-Driven Exploration by Self-Supervised Prediction, *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, JMLR.org, p.2778–2787.

[15] **Bellemare, M.G.**, **Srinivasan, S.**, **Ostrovski, G.**, **Schaul, T.**, **Saxton, D. and Munos, R.** (2016). Unifying Count-Based Exploration and Intrinsic Motivation, *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, Curran Associates Inc., Red Hook, NY, USA, p.1479–1487.

[16] **Ostrovski, G.**, **Bellemare, M.G.**, **van den Oord, A. and Munos, R.** (2017). Count-Based Exploration with Neural Density Models, *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, JMLR.org, p.2721–2730.

[17] **Zhao, R. and Tresp, V.** (2019). Curiosity-Driven Experience Prioritization via Density Estimation, *CoRR*, *abs/1902.08039*, http://arxiv.org/abs/1902.08039, 1902.08039.

[18] **Tang, H.**, **Houthooft, R.**, **Foote, D.**, **Stooke, A.**, **Chen, X.**, **Duan, Y.**, **Schulman, J.**, **De Turck, F. and Abbeel, P.** (2017). #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning, *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, p.2750–2759.

[19] **Choshen, L.**, **Fox, L. and Loewenstein, Y.** (2018). DORA The Explorer: Directed Outreaching Reinforcement Action-Selection, *ArXiv*, *abs/1804.04012*.

[20] **Schlegl, T.**, **Seeböck, P.**, **Waldstein, S.**, **Schmidt-Erfurth, U. and Langs, G.** (2017). Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery, pp.146–157.

[21] **Schlegl, T.**, **Seeböck, P.**, **Waldstein, S.M.**, **Langs, G. and Schmidt-Erfurth, U.** (2019). f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks, *Medical Image Analysis*, *54*, 30–44, https://www.sciencedirect.com/science/article/pii/S1361841518302640.

[22] **Howard, R.** (1960). *Dynamic programming and Markov processes*, Technology Press of Massachusetts Institute of Technology, https://books.google.com.tr/books?id=fXJEAAAAIAAJ.

[23] **Sutton, R.S.**, (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming, **B. Porter and R. Mooney**, editors, Machine Learning Proceedings 1990, Morgan Kaufmann, San Francisco (CA), pp.216–224, https://www.sciencedirect.com/science/article/pii/B9781558601413500304.

[24] **Schmidhuber, J.** (1990). *Making the world differentiable: on using self supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments*, Forschungsberichte Künstliche Intelligenz, Inst. für Informatik, https://books.google.com.tr/books?id=9c2sHAAACAAJ.

[25] **Mozer, M.C. and Bachrach, J.** (1989). Discovering the Structure of a Reactive Environment by Exploration, ***D. Touretzky**, editor, Advances in Neural Information Processing Systems*, volume 2, Morgan-Kaufmann, https://proceedings.neurips.cc/paper/1989/file/1700002963a49da13542e0726b7bb758-Paper.pdf.

[26] **Barto, A.G.**, **Bradtke, S.J. and Singh, S.P.** (1991). Real-time learning and control using asynchronous dynamic programming, **Technical Report**, University

of Massachusetts at Amherst, Department of Computer and Information Science.

[27] **Thompson, W.R.** (1933). On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples, *Biometrika*, *25*(3/4), 285–294, http://www.jstor.org/stable/2332286.

[28] **Weng, L.**, (2020), Exploration Strategies in Deep Reinforcement Learning, https://lilianweng.github.io/posts/2020-06-07-exploration-drl/.

[29] **Schmidhuber, J.** (1991). Curious model-building control systems, *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pp.1458–1463 vol.2.

[30] **Kramer, M.A.** (1991). Nonlinear principal component analysis using autoassociative neural networks, *AIChE Journal*, *37*(2), 233–243, https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209, https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690370209.

[31] **Oord, A.v.d.**, **Kalchbrenner, N.**, **Vinyals, O.**, **Espeholt, L.**, **Graves, A. and Kavukcuoglu, K.** (2016). Conditional Image Generation with PixelCNN Decoders, *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, Curran Associates Inc., Red Hook, NY, USA, p.4797–4805.

[32] **Oh, J.**, **Guo, Y.**, **Singh, S. and Lee, H.** (2018). Self-Imitation Learning, ***J. Dy and A. Krause**, editors, Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, PMLR, pp.3878–3887, https://proceedings.mlr.press/v80/oh18b.html.

[33] **McCulloch, W.S. and Pitts, W.** (1943). A logical calculus of the ideas immanent in nervous activity, *The bulletin of mathematical biophysics*, *5*(4), 115–133.

[34] **Rosenblatt, F.** (1958). The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review*, *65 6*, 386–408.

[35] **Farnia, F. and Ozdaglar, A.E.** (2020). GANs May Have No Nash Equilibria, *CoRR*, *abs/2002.09124*, https://arxiv.org/abs/2002.09124, 2002.09124.

[36] **Radford, A.**, **Metz, L. and Chintala, S.** (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ***Y. Bengio and Y. LeCun**, editors, 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, http://arxiv.org/abs/1511.06434.

[37] **Arjovsky, M.**, **Chintala, S. and Bottou, L.**, (2017), Wasserstein GAN, https://arxiv.org/abs/1701.07875.

[38] **Gulrajani, I.**, **Ahmed, F.**, **Arjovsky, M.**, **Dumoulin, V. and Courville, A.C.** (2017). Improved Training of Wasserstein GANs, *CoRR*, *abs/1704.00028*, http://arxiv.org/abs/1704.00028, 1704.00028.

[39] **Wu, J.**, **Huang, Z.**, **Thoma, J.**, **Acharya, D. and Van Gool, L.** (2018). Wasserstein divergence for gans, *Proceedings of the European Conference on Computer Vision (ECCV)*, pp.653–668.

[40] **Ruder, S.** (2016). An overview of gradient descent optimization algorithms, *CoRR*, *abs/1609.04747*, http://arxiv.org/abs/1609.04747, 1609.04747.

[41] **Ioffe, S. and Szegedy, C.** (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *CoRR*, *abs/1502.03167*, http://arxiv.org/abs/1502.03167, 1502.03167.

[42] **Kingma, D. and Ba, J.** (2014). Adam: A Method for Stochastic Optimization, *International Conference on Learning Representations*.

[43] **Kauten, C.**, (2018), Super Mario Bros for OpenAI Gym, GitHub, https://github.com/Kautenja/gym-super-mario-bros.

CURRICULUM VITAE

Name Surname: Doğay Kamar

EDUCATION:

- **B.Sc.:** 2018, Istanbul Technical University, Faculty of Computer and Informatics, Department of Computer Engineering
- **M.Sc.:** 2022, Istanbul Technical University, Graduate School, Department of Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2019-ongoing Research Assistant at Department of Computer Engineering, Istanbul Technical University

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Kamar D.**, Üre N., Ünal G., (2022). GAN-based Intrinsic Exploration For Sample Efficient Reinforcement Learning, *International Conference on Agents and Artificial Intelligence - ICAART,* Volume 2, 264-272.

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Aktı Ş., **Kamar D.**, Özlü Ö., Soydemir İ., Akcan M., Kul A., Rekik İ., 2022. A comparative study of machine learning methods for predicting the evolution of brain connectivity from a baseline timepoint, *Journal of Neuroscience Methods,* 368, 109475.
- **Kamar D.**, Akyol G., Mertan A., İnceoğlu A., 2020. Comparative Analysis of Reinforcement Learning Algorithms on TORCS Environment, *2020 28th Signal Processing and Communications Applications Conference (SIU),* 1-4.