UNIVERSITY OF CALIFORNIA

Los Angeles

Curiosity-driven Agent

In Sparse Reward Environment

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Computer Science

by

Yaowei Guo

2021

ABSTRACT OF THE THESIS

Curiosity-driven Agent In Sparse Reward Environment

by

Yaowei Guo

Master of Science in Computer Science

University of California, Los Angeles, 2021

Professor Song-Chun Zhu, Chair

In many reinforcement learning scenarios such as many game environments or real life situations, the rewards are usually very limited and sparse. This kind of tasks are always difficult for agent to learn and explore. In fact, dealing with sparse reward environments has always been a challenge in reinforcement learning. In this work, we aim to study the agent driven by curiosity that can handle tasks with sparse rewards using a self-supervised method. We also want to test the possibility about agents driven by their intrinsic curiosity being able to explore the environment and generate reward by themselves. As a result, curiosity makes agents more intelligent. In the later experiments, we test two curiosity based RL methods in three different games and demonstrated that curiosity could indeed achieve very impressive performance in sparse reward game environment.

The thesis of Yaowei Guo is approved.

Yingnian Wu

Demetri Terzopoulos

Song-Chun Zhu, Committee Chair

University of California, Los Angeles

2021

To my family and friends.

Thank you for all your support and confidence.

TABLE OF CONTENT

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank professor Song-Chun Zhu as being the committee chair and supervised my thesis. During my graduate study, I have learned a lot from professor Zhu's lectures.

I would like to thank professor Yingnian Wu. His lectures brought me into the field of machine learning. The great amount of knowledge he taught me in reinforcement learning area is crucial for me to accomplish this work. Without professor Wu's careful instruction, it is impossible for me to finish my thesis.

I would like to thank professor Demetri Terzopoulos. The knowledge I learned from his lectures also helps me a lot when writing up this work. It has been a great honor for me to have the opportunity to become a student of these preeminent professors.

I would like to thank Xu Xie as being my mentor throughout my research life. Xu taught me how to think and work as a researcher through tremendous conversations I had with Xu. It has been a great honor for me to work with Xu during my graduate study.

This draft of this work is based on a project I did with Victor Zhang, Ziheng Xu and Ximing Chen. I would like to thank to their contributions towards formulating the problem and conducting the experiment. The discussion we had helped me a lot when accomplishing this work.

CHAPTER 1

Introduction

In these years, research on reinforcement learning problem has achieved very impressive results in training agents to accomplish a variety of complex tasks, especially in environments that can provide abundant and uninterrupted rewards for agents to learn and maximize. This kind of task can be found in many Atari games and robot controlling problems [1, 2]. In general, agent dealing with this kind of problem can model the environment as a Markov Decision Process(MDP), and then manage to update policies to maximize the long-term reward given by the environment. However, in many games and real-life environments, the reward function given by outside environment is very sparse, completely non-existent, or needs to be carefully designed by researchers. In this case, the agent could hardly update the policy without receiving any feedback. On the other hand, it is always not feasible or scalable for researcher to manually craft enough reward for agent in those environments since the amount of reward that agent needs in order to explore environment and update policy is often tremendous. As a result, designing agent that can handle sparse reward environment has always been a problem in reinforcement learning.

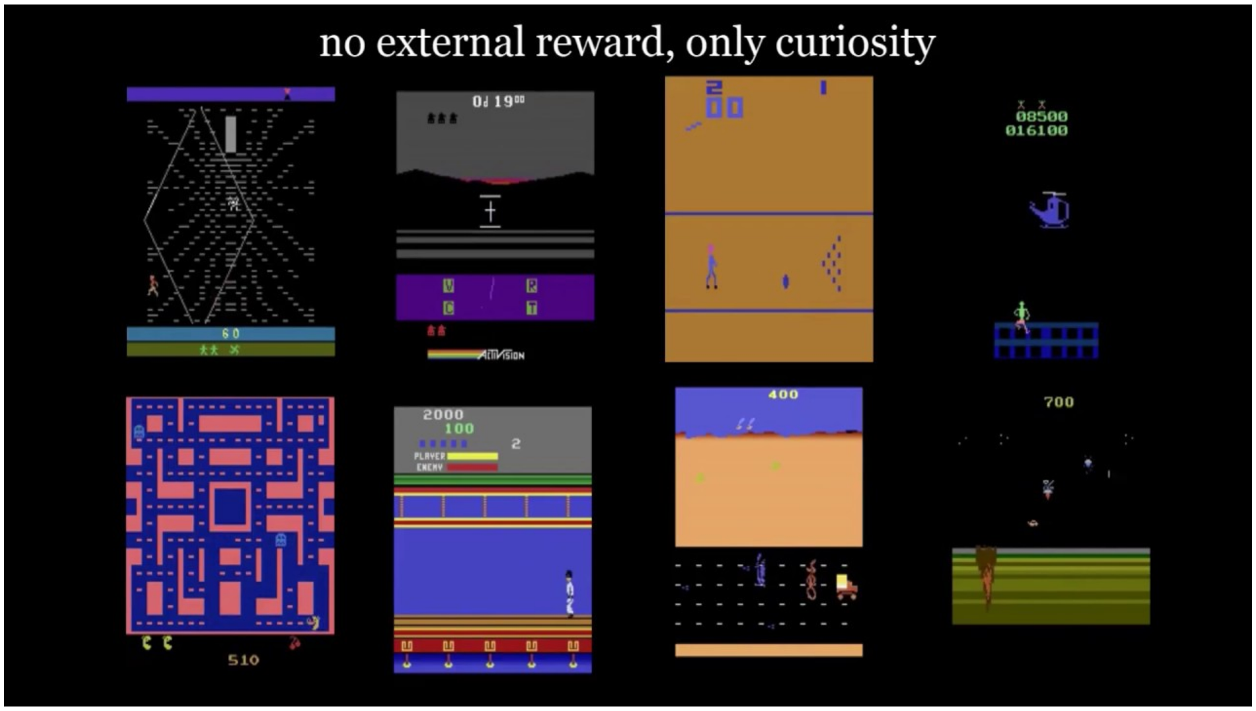

Figure 1.1: Some RL tasks that have very limited or even absent extrinsic reward. Agent in these tasks treats curiosity as intrinsic motivation and tries to solve the problem only with intrinsic reward. In later chapter we will see that curiosity-driven agent could achieve very impressive results in sparse reward environments.

To help solve this problem, we want to investigate the situation where curiosity and diversity functions are used as the intrinsic rewards and drives the agent to gain better perception of the RL problem. Figure 1.1 shows many situations that curiosity-driven agent only uses intrinsic reward to explore novel environments. In addition, we also want to investigate the effectiveness about how agent driven by curiosity gets familiar with and explores the environment. Virtual agents can thus be trained in new environments in a self-supervised way, and does not require hand-crafted reward functions.

The idea of using curiosity to learn is not a new idea [3]. In fact, the real world that human lives in are such a sparse reward environment that human needs to discover new stuffs and adapt new knowledges in every minute. Developmental psychology researches have long proposed curiosity as a crucial motivation for humans learning and exploring new environments [4], especially in young children. Young children actively learn about the world by exploring it, without specific task or goal in mind. The commonsense skills developed by young children are important for many tasks later on. However, the lack of these commonsense skills is one of the biggest roadblock for current AI systems.

In this work, we test the performances of two curiosity based RL algorithms: Intrinsic Curiosity Module(ICM) and Random Network Distillation(RND) on three sparse reward game environments: Atari Breakout, Montezuma‘s Revenge, and Super Mario Bros with two kind of inputs: image input and Atari Simulator RAM input respectfully. In ICM, agent mainly learns from intrinsic reward to update, and seldom receives any extrinsic reward from outside environment. On the other hand in RND, agent receives both intrinsic and extrinsic rewards although extrinsic rewards may always absent, and the total reward is the weighted average of these two. We compare the results of both ICM and RND methods and show that 1) intrinsic reward generated by curiosity helps agent to explore the environment in sparse reward problem. 2) The use of RND method has positive effect on performance than vanilla ICM method.

CHAPTER 2

Related Works

The study of using intrinsic motivation to drive agent to explore environment is not a new concept in reinforcement learning [5, 6]. According to researchers, there are two main advantages of agent driven by intrinsic motivation: firstly, agent driven by intrinsic reward tend to discover new states during actions [11, 12]. Secondly, agent would try to move to states that could decrease the error since agent is able to predict the environment state change to a certain extent after it takes actions [13].

Curiosity as a typical class of intrinsic motivation has also been studied to solve sparse reward problem by many researchers [7, 8]. These works show that curiosity serves as an important factor for agent to get familiar with environment. In addition, curiosity could also helps agent adapt to new environment faster after agent had gained some perception to the task. This has been shown in many game scenarios that agent is able to learn faster in new level after it completes previous level.

In addition to curiosity, other types of intrinsic reward of agent such as surprise and diversity are also proposed in other works [9, 10]. In these works, agent learn variety of different policies by exploring different parts of environments. Therefore when agent encounter novel scenarios later, agent could reuse the set of policies already learned to accelerate the training process in new scenarios.

Two methods that are used in curiosity-driven agent are Intrinsic Curiosity Module(ICM) [7] and Random Network Distillation(RND) [14], which we are going to discuss in this work. In general, total reward of the agent is the weighted sum of both intrinsic and extrinsic rewards; however, extrinsic rewards would always be zero in sparse reward environment. Agent will try to maximize the long term total reward while curiosity encouraging agent to reach new states throughout exploration.

CHAPTER 3

Algorithm and Methodology

Before we step into the discussion of curiosity-driven method, let us first briefly review the format of reinforcement learning problem. In principle, at each time step t, agent interacts with environment to get a current state s_t, an action a_t, and a reward r_t. Agent will then try to perform the next available action a_{t+1} to reach new state s_{t+1} and receive new reward r_{t+1} at next time step. This action a_{t+1} was determined based on the learned policy π, which is a conditional distribution over the set of actions A given the set of states S. Therefore the goal of RL algorithm is to learn a set of policies π that maximize the cumulative reward.

The idea of using curiosity to training is not new. However, there has been few computationally feasible models. Under the MDP and RL framework, the goal of exploration is to learn the transition model of the world $P(s_{t+1} \mid s_t, a_t)$. To courage agent exploring novel states, we generate large intrinsic rewards for unexplored states, and train existing RL frameworks with a weighted average of the intrinsic rewards and extrinsic rewards. In the rest of this section, we will discuss several relevant framework proposed in literature.

3.1 Intrinsic Curiosity Module

One framework of curiosity-based intrinsic reward is the Intrinsic Curiosity Module (ICM) proposed by [7]. A very simple architecture of ICM is shown in Figure 3.1. When dealing with the problem, curiosity-driven agent not only receives the intrinsic reward generated by ICM, but also may receives extrinsic reward to a certain extent from outside environment occasionally. Therefore the total reward will be the sum of both intrinsic reward and extrinsic reward over time *t*:

$$R(t) = \sum_{t=1}^{T} (r_t + r_t^i) \tag{3.1}$$

In this equation, $R(t)$ is the total reward, r_t is the extrinsic reward provided by environment, and $r_t{}^i$ is the intrinsic reward generated by ICM. It is also worth to note that most of the time r_t would be zero in sparse extrinsic reward problem. In fact, when we train agent with ICM in next section we will show that during learning process cases where r_t is not zero is so rare that agent actually learns only from $r_t{}^i$. Nevertheless the agent is still able to achieve handsome result without extrinsic reward.

According to [7], at each time step *t*, agent will try to perform action a_t chosen from the policy π. The goal of agent is to learn the policy that maximize the expected total reward over time *t* through following equation:

$$\max_{\theta_P} \mathbb{E}_{\pi(s_t,\theta_P)}[R(t)] \tag{3.2}$$

where s_t is current state and θ_P is the parameter of policy π which we want to optimize.

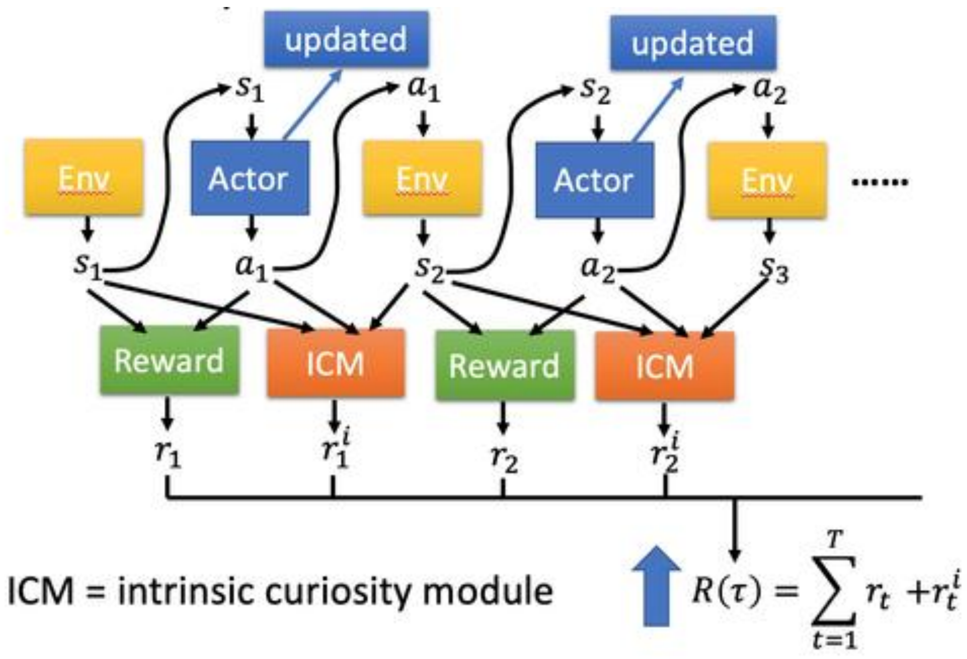

Figure 3.1: A graph that shows the very basic structure of ICM. In the figure, state s_t and action a_t are inputs at time t. r_t is the reward provided by environment at time t, and $r_t{}^i$ is the reward generated by Intrinsic Curiosity Module(ICM), a curiosity based reinforcement learning method which we are going to discuss in this chapter. $R(t)$ is the goal of algorithm that maximizes the sum of r_t and $r_t{}^i$.

In order to courage agent to explore novel states, one intuitive method is to give states that are hard to predict higher intrinsic reward. In other word, we want to tie a direct proportion between $r_t{}^i$ and prediction error:

$$r_t^i \propto \hat{s}_{t+1} - s_{t+1} \tag{3.3}$$

However, not all of the unpredictable states are important to agent. For example the changes of background cloud in Super Mario game, although hard to predict, is irrelevant with agent's movement, which does not worth agent to pay much curiosity. As a result, how to make sure prediction error worth agent's curiosity is a question that need consideration.

In ICM, curiosity is formulated as the prediction error of the next state in the feature space according to [7]. $\hat{\phi}(s_{t+1})$, which is the prediction of feature space of state s_{t+1}, is computed by:

$$\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t; \theta_F) \tag{3.4}$$

giving current feature space $\phi(s_t)$, action a_t , parameter θ_F as inputs and computed using forward dynamics model f. θ_F is learned to minimize the loss of prediction error $||\hat{\phi}(s_{t+1}) - \phi(s_{t+1})||_2^2$.

$r_t{}^i$ is computed by the MSE of prediction loss:

$$r_t^i = \frac{\eta}{2}||\hat{\phi}(s_{t+1}) - \phi(s_{t+1})||_2^2 \tag{3.5}$$

where $\eta > 0$ is a scaler and ϕ denotes the mapping from raw sensory space to feature space. The reason for mapping the observation state to the feature state is that prediction error on raw sensory data for example pixels is highly subject to the the stochastic nature of the environment. Feature space is a layer of abstraction that can help reduce the noise.

The particular features chosen in ICM is the inverse dynamics model g. The mapping is trained with a separate task of predicting the current action a_t, given the current state s_t and the next state s_{t+1}:

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I) \tag{3.5}$$

[7] introduces parameter θ_I which tries to minimize the loss between $\hat{a}_t$ and a_t. We want our predicted $\hat{a}_t$ to be close to a_t since it indicates that feature space are highly predictable and are related to the agent's action. On the other hand, large difference between $\hat{a}_t$ and a_t probably

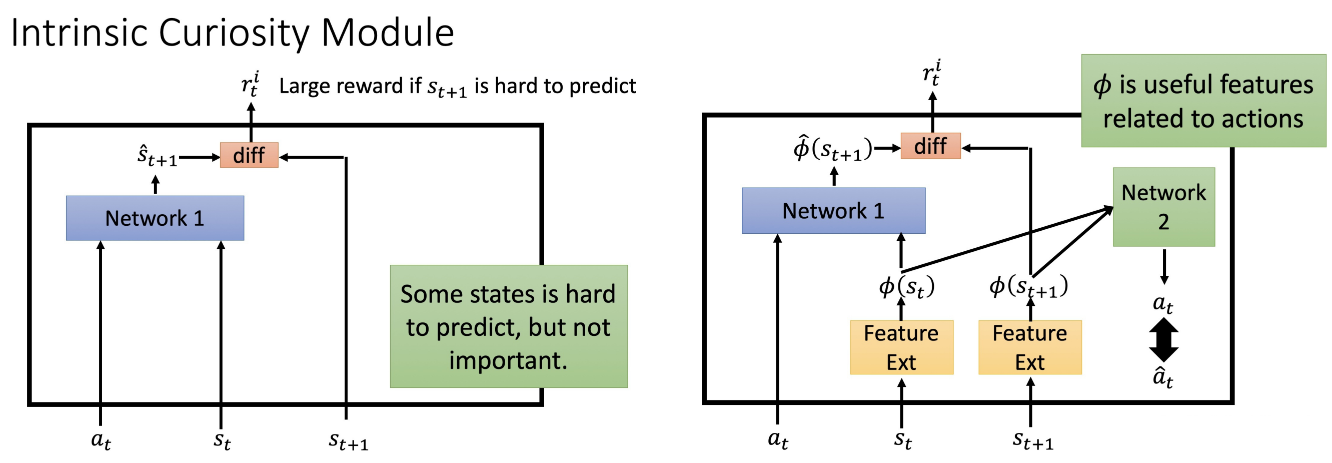

Figure 3.2: while the agent is interacting with the environment, an inverse dynamics model is trained on the task of predicting the action of the agent a_t given the current state s_t and the next state s_{t+1}. This inverse dynamics model learned to effectively encode the states into feature vectors, which is of much lower dimension.

illustrates that the state change after performing action is unpredictable, perhaps the environmental changes. The goal of this inverse dynamics task is to reduce the influence of the elements in the environment that are not very relevant to the agent's actions. See Figure 3.2 as illustration about determining meaningful features.

The combination of forward and inverse dynamic models make sure that agent only curious about the environment states changes which are relevant to its actions, instead of paying attention to other changes such as background alternations which do not affect agent's action. The goal of ICM is to learn the three parameters mentioned above: θ_P, θ_F, and θ_I. θ_P tries to maximize the expected total reward through policies, while θ_F and θ_I try to minimize the loss of forward and inverse dynamic models. In next section, we will discuss our experiments with ICM method on three RL game environments.

3.2 Random Network Distillation

As mentioned above, ICM tends to pay more curiosity on next states that are novel and hard to predict. Therefore ICM might encounter problems in environments that always change dramatically. One problem of the ICM framework is the "noisy-TV problem". While exploring the world, if the agent encounters a TV is randomly changing channels, the agent would stay in front of the TV and not explore other parts of the world, because the next state of this stochastic changing TV is very difficult for agent to predict, therefore curiosity for the content on the TV generates a very large intrinsic reward for the agent.

One framework proposed to alleviate this problem is the Random Network Distillation (RND) framework proposed by [14]. The overall formulation is the same as ICM, but instead of using inverse dynamics features, RND uses random features. The architecture is quite simple. A target network that encodes the feature space information of states is randomly initialized in the beginning of the training process, and its weights are never updated. A second network called the prediction network is trained over time to approximate the target network, and the prediction error is used as the intrinsic reward for the agent.

Because the target network stays the same, the feature space information generates from target network will also stay the same. If the agent has explored a certain state, the prediction network has seen the output of the target network. The prediction error for this explored would be small, and the intrinsic reward would be small as well. For unfamiliar states, the prediction error and intrinsic reward would be large. Figure 3.3 [15] shows the overall architecture of target network and prediction network.

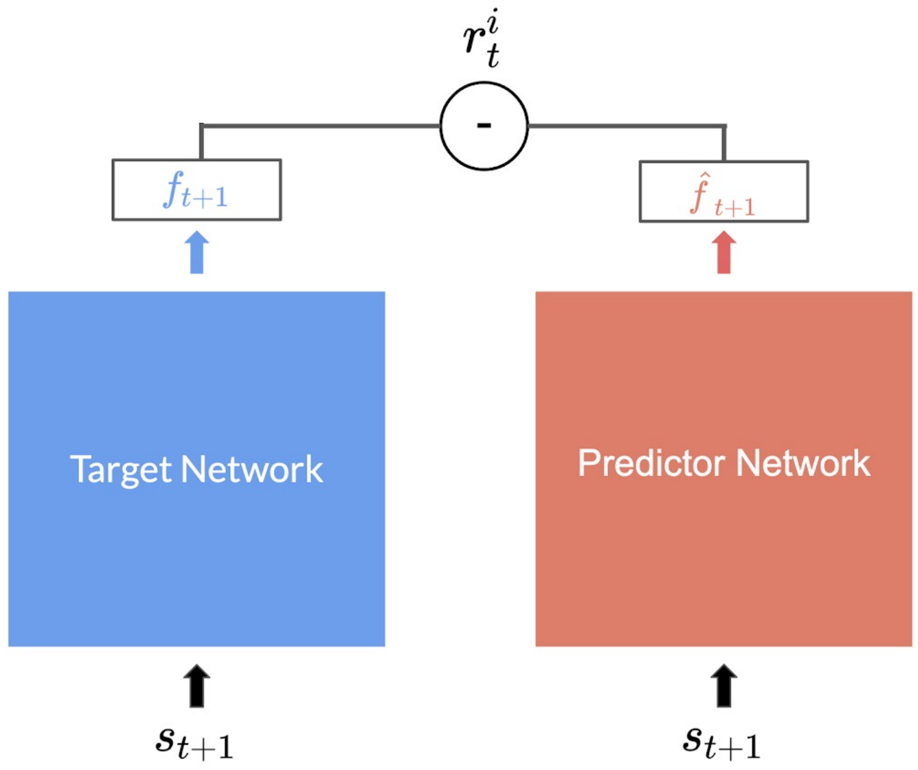

Figure 3.3: Graph that shows the structure of target network and predictor network. Target network which encodes feature space information of all states is randomly initialized and never update. It takes next state s_{t+1} as input and output corresponding feature space f_{t+1}. Predictor network is trained to approximate target network. It takes s_{t+1} as input and tries to predict the target network's output $\hat{f}_{t+1}$. Intrinsic reward $r_t{}^i$ will be the MSE of f_{t+1} and $\hat{f}_{t+1}$.

RND then calculates the intrinsic reward $r_t{}^i$ as the MSE of f_{t+1} and $\hat{f}_{t+1}$:

$$r_t^i = ||\hat{f}_{t+1} - f_{t+1}||_2^2 \tag{3.6}$$

while predictor network will try to minimize the loss of this prediction error.

The behind logic of both ICM and RND are same: both algorithms tend to encourage agent to explore novel states. They pay much curiosity and give large intrinsic rewards to states that have not visited before and are hard to predict, while only putting small intrinsic reward on previously visited states which are highly predictable. However, unlike ICM, the predictor network of RND is trained to predict the feature space information from target network instead of directly predicting next state. Therefore RND can make sure it gives higher intrinsic reward to state that its feature space information is meaningful, no matter whether the state itself is predictable or not. Through minimizing the prediction loss, the predictor network can better approximate the feature space information from target network and pay less curiosity to those “noisy states”. Therefore RND is able to alleviate the effect of noisy-TV problem. In next section, we will present our experiment results with RND method on three RL game environments and compare the outputs with that of vanilla ICM method.

3.3 Diversity

Another related framework is proposed by [10] that called Diversity is All You Need(DIAYN) which uses diversity as intrinsic motivation for agent. DIAYN connects reinforcement learning with information theory, and enables intelligent virtual agents to develop skills without an external reward function. New skills emerge from maximizing an information theoretic objective. In this work, the agents are first trained without any task to learn a set of skills. Each skill is essentially a different policy. The agents try to develop a set of skills, where each skill is as different as possible from others by exploring drastically different parts of the state space. Then, for different tasks, the set of skills can be reused to accelerate training. In addition, developing diverse skills can also help agent to distinguish and avoid noisy states, which makes skills more useful.

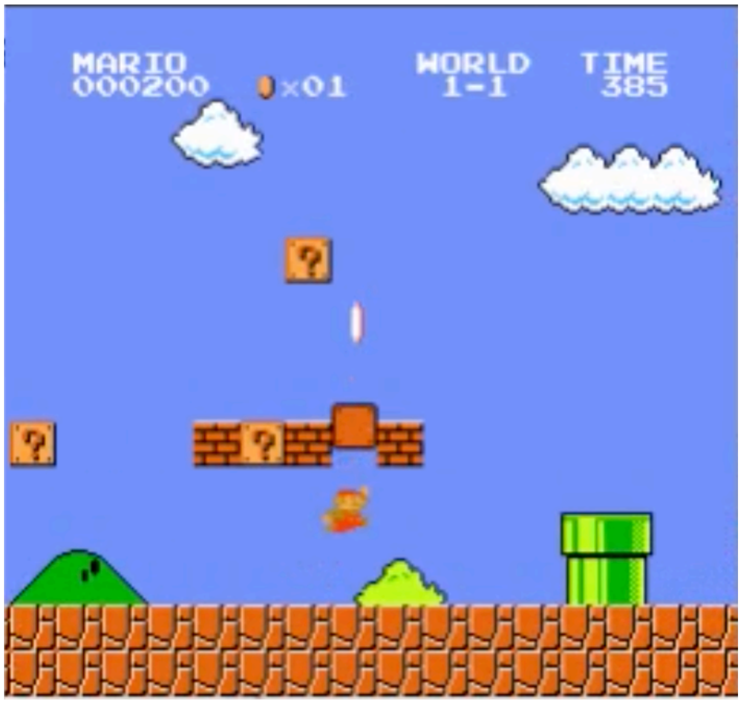

Figure 3.4: Screenshot that shows DIAYN's ability of imitating human performance. When DIAYN playing Super Mario Bros, agent is able to develop skill "jump" in order to collect coin like human. Although "jump" is not a necessary skill at current position, DIAYN also considers this skill as useful.

One main advantage of using diversity as intrinsic reward is to imitate experts. Figure 3.4 shows this feature when agent learns to play Super Mario. Just like human player, agent learns to "jump" under the coin block to collect coin although this action does not directly help agent solve the level.

CHAPTER 4

Experiments and Results

As we discussed in previous section, we experimented both ICM and RNG algorithms in three different game environments: Atari Breakout, Montezuma, and Super Mario Bros. In order to compare the performance of these two algorithms, we have two different type of inputs for the first two games: the classic image input, and the Atari Simulator RAM input.

The model for image input is much more complicated, and training each episode is slower. The models that use Atari Simulator RAM as input is much simpler, since the RAM is only 128 bytes, but we need to design and tune each model. Some hyper parameter we use in both ICM and RND are learning rate being $1e-4$, γ being 0.99, and λ being 0.95 with batch size equals 128. In most of the experiments we get handsome results after we train the agent within 1K episodes. In ICM we use a three convolution layers network with channel sizes equal 32, 64, 64, kernel sizes equal 8, 4, 3, and strides equal 4, 2, 1 respectively. After that, we add a fully connected layer to connect feature size with output size. In RND we use network structure of 4 layers fully connected network, with dimension: state size -> 256 -> 256 -> 256 -> action size, where state size and action size are the dimensions of each state and action. Each layer we use a

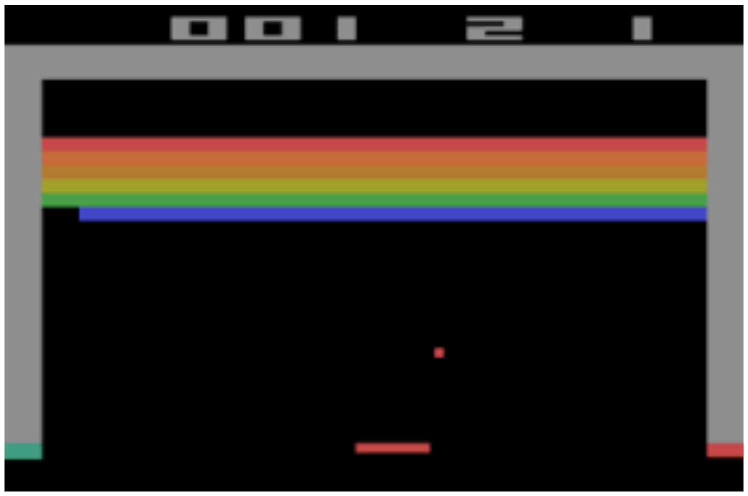

Figure 4.1: Breakout uses either an array of shape (210,160,3) RGB image or 128 bytes RAM of Atari machine as the game screen. Every action is repeatedly performed for a duration of *k* frames, where *k* is uniformly sampled from {2,3,4}. Player controls the horizontally movable paddle to avoid the ball from touching the bottom of the screen and tries to hit the bricks as much as possible.

linear transformation on inputs. For the forward activation, we use ReLU to build the network that maps state to action.

4.1 Atari Breakout

We start with using two algorithms to play Atari Breakout, shown in Figure 4.1. Atari Breakout is a very classic arcade game in RL. Compared with later two RL games, Atari Breakout can provide explicit and dense extrinsic reward as feedback to agent to guide its future movement. Therefore for experiment results we would like to pay more attention on the intrinsic reward generated by two algorithms, and how agent explores environment driven by intrinsic reward. The control of this game is relatively simple since it only supports two directions as agent's action in order to move paddle horizontally.

For Breakout, we run the agent with 16 parallel environment to avoid spending numerous time on training. We set the maximum steps per episode to 4500, which means the agent can perform as much as 4500 actions before it dies in each episode. This mainly prevents agent stuck in irrelevant states forever such as encounters Noisy-TV problem. For our expectation, we would like to see agent survive as long as possible as well as maximizing the score.

4.1.1 Breakout's RND Atari Simulator RAM Input

We first run RND algorithm on Breakout game with RAM input for 1500 episodes. The results are shown in Figure 4.1.1. We can observe that the intrinsic reward reaches its maximum in first several episodes, and becomes smaller and smaller as episode increases. While the intrinsic reward tend to vanish around 800 episodes, and does not move much after that. We think the reason is that the game environment of Breakout is relatively simple. There are only two actions that agent can take, which indicates that few episodes is enough for agent to get familiar with game environment. However, this illustrates that RND method learns very fast as well, and tends to explore new states rapidly. It is also worth to note that RND method is able to distinguish noisy states since intrinsic reward decreases after agent has explored the entire game environment.

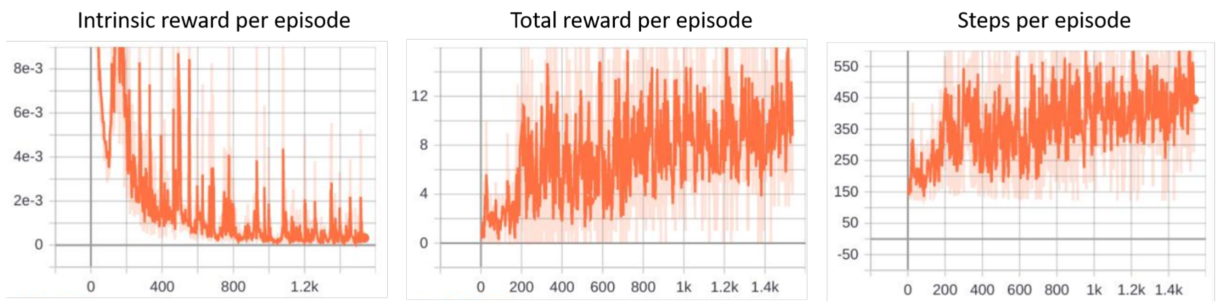

Figure 4.1.1: The result of playing Breakout game using RAM input. First two graphs show the reward per episode and total reward for 1500 episode, and the last graph shows the number of action that agent takes per episode.

The second graphs shows the total reward per episode of agent. We can observe that total reward increase linearly in general; however, it exhibits the high-variance, even after 1000 episodes the learned policy can be unstable. As we mentioned before, Breakout can provide efficient extrinsic reward, which explains linear growth of total reward. The last graph shows how many movements the agent takes in each episodes. Since the agent is able to perform one movement every four frames, it also indicates the how long does the agent survive in each episode. We can observe that in general the step number increase as episode number increase, which illustrates the success of RND algorithm in Atari Breakout game.

If we connect the first graph with the third graph. We could see that intrinsic reward nearly disappears after 1000 episodes. Meanwhile the step number that agent performs also stop increasing after 1000 episodes. This emphasizes that intrinsic reward plays a crucial role for agent's survival. As a result, after training for 4 hours, RND is able to captures 52 score in first level. We believe that RND method can achieve better result if we keep our training.

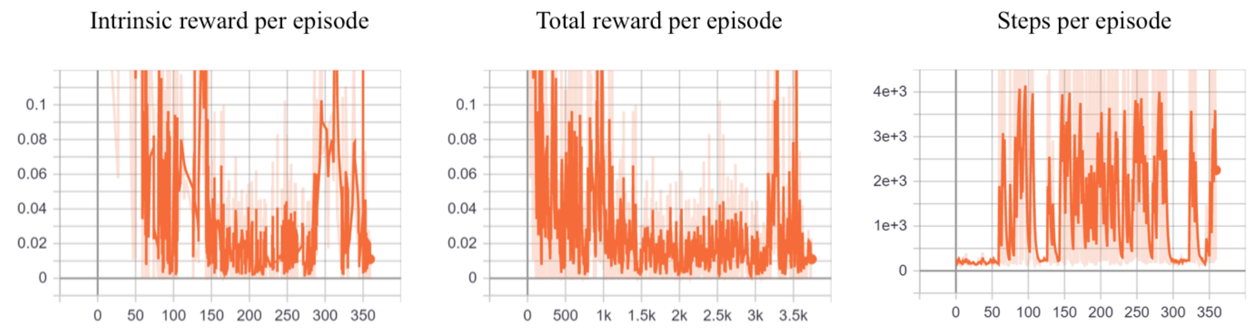

Figure 4.1.2: The result of playing Breakout game using Image input. First two graphs show the reward per episode and total reward for 350 episode, and the last graph shows the number of action that agent takes per episode.

4.1.2 Breakout's ICM Image Input

We then run ICM algorithm on Atari Breakout game with image input for 350 episodes. The results are shown in Figure 4.1.2. The first two graphs, we can observe that intrinsic reward shows an U shape: it decreases from its first high peak in first 150 episodes; then it drops into a valley and vibrates around 0.02 in next 100 episodes; in the last 100 episodes the intrinsic reward starts to increase to the second high peak. Overall, the variation of the reward is very large(from above 0.1 to almost 0).

However, the performance of ICM method differs from that of RND method dramatically. If we compare first graph of two methods, we can observe that intrinsic reward generated by ICM is much higher than intrinsic reward of RND(0.1 vs. 0.008). The discrepancy between values of intrinsic rewards may also tell the difference between two methods about how they calculate intrinsic reward. We now know that ICM tends to provide much more intrinsic reward to agent than RND, and it is no doubt that agent driven by ICM can learn and explore faster giving more abundant intrinsic reward. As a result, ICM is able to capture 33 score after only training 350 episodes, which is much better than RND from the output score's point of view(remember that RND achieves 52 score after training 1500 episodes). The reason of why we stop training at 350 episodes is that training ICM requires much more time than training RND. In fact, training ICM for 350 episodes already takes us 4 hours. From the training time's point of view, RND outplays ICM. The tradeoff between output score and training time need careful consideration when selecting algorithm.

It is also worth to note that ICM hardly learns from extrinsic reward. This can be seen from the second graph of Figure 4.1.2 that the difference between total reward and intrinsic reward of ICM can be negligible. The reason might be that ICM focuses on sparse reward environment where extrinsic reward is always missing. We believe that this phenomenon will only appear in Breakout game. In later two more complex game environments where extrinsic reward is sparse, total reward of both algorithms will be very close to intrinsic reward, and agent will only learn from intrinsic motivation.

From the third graph we can observe that the step count of ICM is very high. Agent in many episodes performs thousands of actions, which almost reaches the maximum 4500 steps per episode. This might also explains the long training time of ICM. We guess the reason of this high step count is that the simulator may be unresponsive during some episodes the game, and caused the step count to be very high; however, one other explanation is that agent might encounter Noisy-TV problem in the game and sticks in certain states.

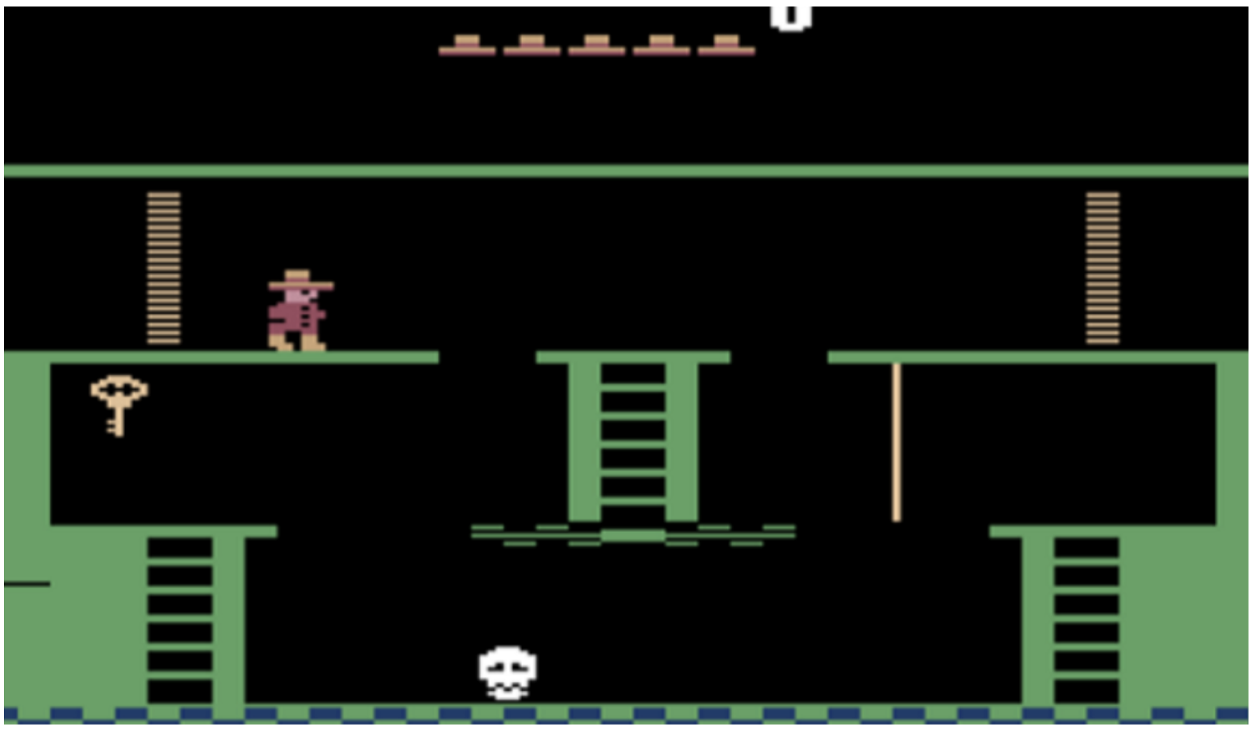

Figure 4.2: Same as Atari Breakout, Montezuma uses either RGB image or Atari RAM as screen. Agent controls the character to reach the key position, while avoid character touching traps and enemies.

4.2 Montezuma‘s Revenge

Unlike Atari Breakout, the game environment of Montezuma is much more complex. Montezuma is a very classic hard exploration problem in RL, and there are a number of papers experiment on Montezuma. This game has sparse extrinsic reward, and the number of action that agent can perform exceeds two. The screen of Montezuma is shown in Figure 4.2. A feature that Montezuma differs from other games is that the game environment of Montezuma could switch entirely by agent entering different rooms. The number of rooms that agent entering during the game has always been an important measurement of agent’s ability about exploring new environment, and has been studied by many researchers. In our experiment, we are also interested in the ability of ICM and RND exploring novel states.

Agent need to collect 23 keys which are distributed in different rooms in total to get into next level, which means during exploration, agent will not receive any extrinsic reward before it reaches any key position. Therefore intrinsic reward plays a significant role for agent learning to play Montezuma. In this game, we run the agent with 64 parallel environments and still set the maximum steps per episode to 4500.

4.2.1 Montezuma RND Atari Simulator RAM Input

We first run RND algorithm on Montezuma game with RAM input for 70 episodes. The results are shown in Figure 4.2.1. From the first graph we can observe that the intrinsic reward decreases dramatically in first 5 episodes; however, it then begins to increase linearly in later episodes. We believe the intrinsic reward will still continue to grow after 70 episodes. We then look at the third graph, the step number per episode of RAM input version shows an unusual sharp peak in first 20 episodes, and then it starts to vibrate around 1300 in the remaining episodes. These results show that the increase of episode number does not connect directly with agent's survival time; however, after training more episodes agent is able to achieve higher score through same number of actions. The linear growth of intrinsic reward of first graph tells us that agent learns from intrinsic reward and explores new rooms continuously. As a result, the performance of RND is very impressive. We observe that agent explores 12 different rooms and

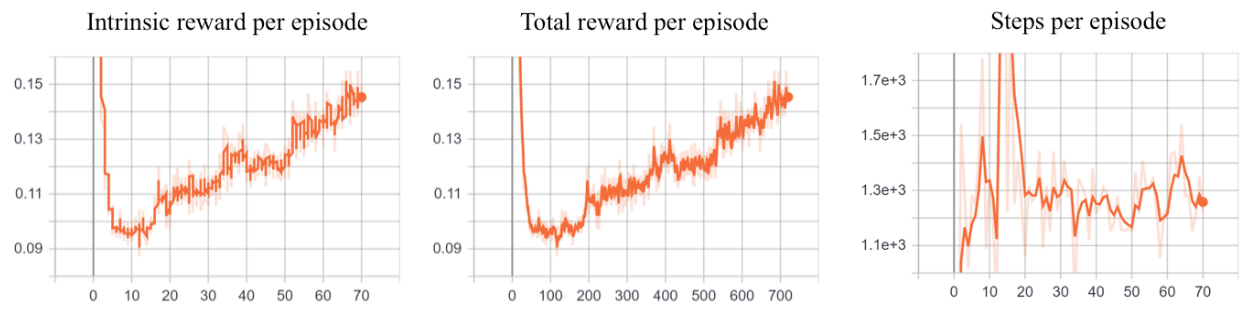

Figure 4.2.1: The result of playing Montezuma game using RAM input. First two graphs show the reward per episode and total reward for 70 episode, and the last graph shows the number of action that agent takes per episode.

collects 6 keys robustly and smoothly in No.70 episode. The actions of RND method is very close to human player. Agent shows a very strong sense of purpose when traveling around different rooms, as well as behaves strategically when collecting keys and avoiding enemies.

We can observe that the difference between intrinsic reward and total reward is negligible from the second graph, which is coherent with our expectation that extrinsic reward being absent. This illustrates that agent is able to achieve very handsome result with only intrinsic reward in sparse reward environment. In fact, we spend 5 hours training the agent for 70 episodes; however, agent manages to capture 7100 scores after only 70 episodes, which is better than a number of human players. We absolutely believe that RND has the ability to capture higher score if we continue the training process.

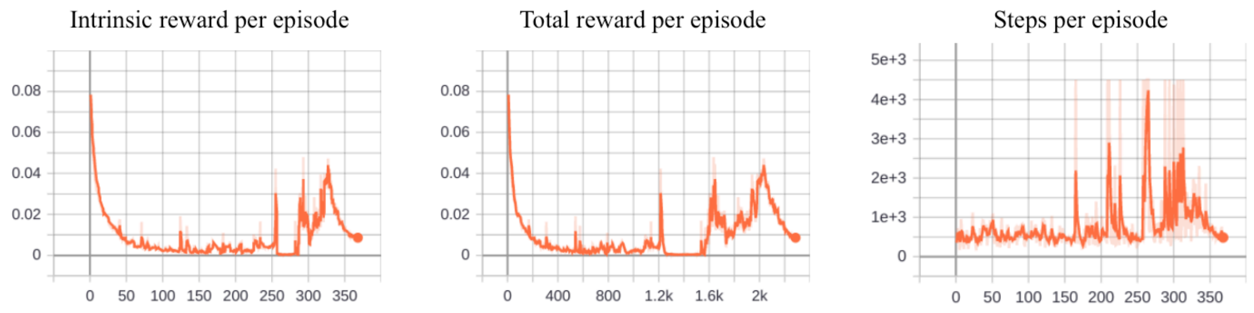

Figure 4.2.2: The result of playing Montezuma game using image input. First two graphs show the reward per episode and total reward for 380 episode, and the last graph shows the number of action that agent takes per episode.

4.2.2 Montezuma ICM Image Input

We then run ICM algorithm on Montezuma's Revenge with image input for 380 episodes. The results are shown in Figure 4.2.2. We can observe that the results are entirely different from that of RAM input. The highest peak of intrinsic reward is at episode 0, and it decreases dramatically to 0 in around 100 episodes. However, the intrinsic reward starts to increase after 250 iteration, then begin to vibrate between 0 to 0.04. In addition, the step graph of Montezuma also seems entirely different from that of Breakout game. The step number remains around 500 in first 150 episodes, and then starts to jump sharply to some unusual high peak occasionally. This indicates that agent performs much more actions at some episodes than others.

If we compare the intrinsic reward of ICM with that of RND from the first graph, we can see that RND provides much higher intrinsic reward. Even the lowest point of RND is higher than the highest point of ICM(0.09 vs. 0.08). In addition, from 100 to 250 episodes, the intrinsic reward of ICM drops nearly to 0. In this case that agent could hardly receive extrinsic reward, which can bee seen from the second graph, we think that agent drive by ICM learns and explores much slower than agent driven by RND. As a result, we only observe that agent explores 5 different rooms and collects 2 keys before it dies in No.380 episode. It is also worth to mention that instead of moving purposeful like RND, agent driven by ICM moves randomly in each room. It spends nearly 10 seconds jumping up and down in first room, which is difficult for us to find out the goal of agent's movement.

The steps per episode graph in Figure 4.2.2 also explains the difference between ICM and RND. ICM takes around 600 steps in average in each episode, which is much fewer than the average steps number of RND(RND takes around 1200 steps in average). Fewer steps number implies not only that agent drive by ICM lives shorter, but also that agent discovers less rooms during the game. We also guess that agent might encounter Noisy-TV problem during training since the step number is extremely high at some episodes.

Admittedly, Montezuma's Revenge is a famous hard exploration problem in RL. Agent has to discover the key positions by it self through exploring different rooms. We train ICM for 11 hours, and ICM is able to capture 2500 scores in 380 episode. Although much lower than RND, the performance of ICM is still acceptable.

4.3 Super Mario Bros

The third game environment of our experiment is the famous Super Mario Bros. The screen of Mario is shown in Figure 4.3. Like Montezuma, Mario also has complex roles and environments. Mario game has a number of items, traps, and enemies that could interact with

Figure 4.3: Super Mario Bros only takes image as input. Mario's game environment is much more complicate than previous two games. Agent will need to control the character to discover and reach the goal position, while interacting with other items, traps, and enemies during the game.

character, which means the agent has to distinguish wither a state is useful or irrelevant for completing current level. In addition, unlike previous games, the goal will not be displayed in the game screen when the game starts. Agent need to move forward to discover the goal itself. The action that agent can perform may also change as agent interacts with different type of items.

As we mentioned before, there are a number of actions that are not indispensable for agent and does not help agent complete the level directly such as collecting coins; however, we would still like to courage agent to perform these actions. In consideration of the complex nature of Super Mario Bros, we do not put any extrinsic reward in game environment. Agent will need to learn to play the game from intrinsic reward only. In the other word, curiosity is the only weapon for agent to survive. For game setting, we run the agent with 32 parallel environments and increase the maximum steps per episode to 30000 due to the difficulty of this game.

4.3.1 Mario Using ICM Method

We first let the agent learn the game using ICM algorithm. After more than 900 episodes, agent still stuck at the first level. The results of intrinsic reward and steps number are shown in Figure 4.3.1. We can observe that the results are very different from previous two games. The intrinsic reward quickly drops to 0 in the first 50 episodes, and remains at 0 all the way to almost the end. On the other hand, third graph shows that in most episodes, agent does not perform many useful movement and die very rapidly; however, it seems that agent suddenly knows how to "play" this game in the last 20 episodes although it does not move far from the beginning position. This unusual phenomenon does not appear in previous two games.

In reality, we think that ICM method may not be a proper algorithm for Super Mario Bros. The first graph has clearly shows that ICM actually does not provide enough curiosity and intrinsic reward for agent to learn to play this game. Agent prefers jumping up and down instead of moving forward during training. Even if in some episodes that agent starts to move forward, agent seems to be unaware of any other items in game environment. Agent does not try to interact with coin blocks, does not take any actions to kill or avoid enemies and obstacles, and does not know to jump intentionally to cross gaps. The only thing that agent does is jumping all the time unintentionally, since jumping is the only actions that agent could perform instead of

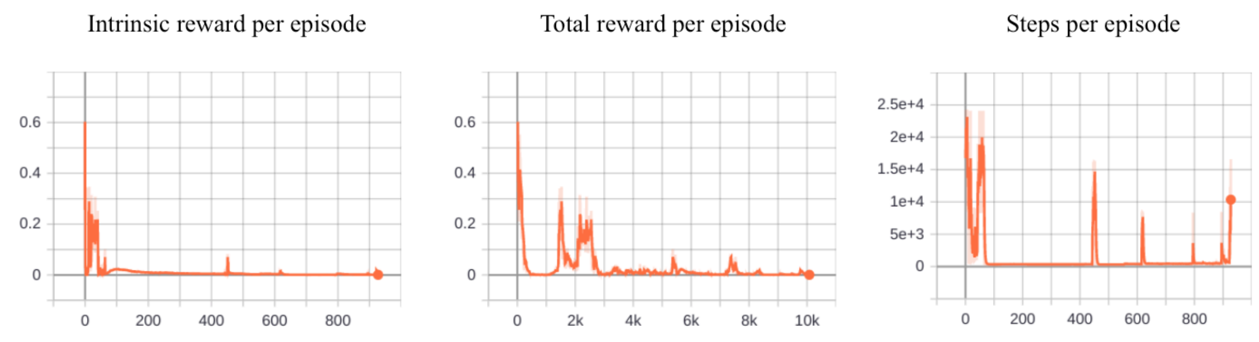

Figure 4.3.1: The result of playing Super Mario Bros game in ICM algorithm. First two graphs show the reward per episode and total reward for 900 episode, and the last graph shows the number of action that agent takes per episode.

moving forward. We think the reason of this poor performance is that the game environment of first level does not provide enough curiosity that motivates agent to explore. In Montezuma, the size of each room is relatively small, and agent could easily explore new rooms driven by its curiosity. On the other hand, in Super Mario, we only have one big "room" in each level so that agent need to keep moving forward in order to generate extra curiosity.

We think that in the early stage of training, agent has indeed tried to move forward to explore new environment. This has been proved in Figure 4.3.1 that both intrinsic reward and step number of first 50 episodes are high enough to generate curiosity. However, after killed by enemies and gap or realizing that interacting with collectable items does not help agent to reach the target, agent might think moving forward is not the right method and stop performing this action, which falls in a vicious circle that agent will not receive any intrinsic reward and curiosity by not moving forward. In conclusion, the logic and structure of ICM need to be further improved in order to achieve better result in Super Mario Bros.

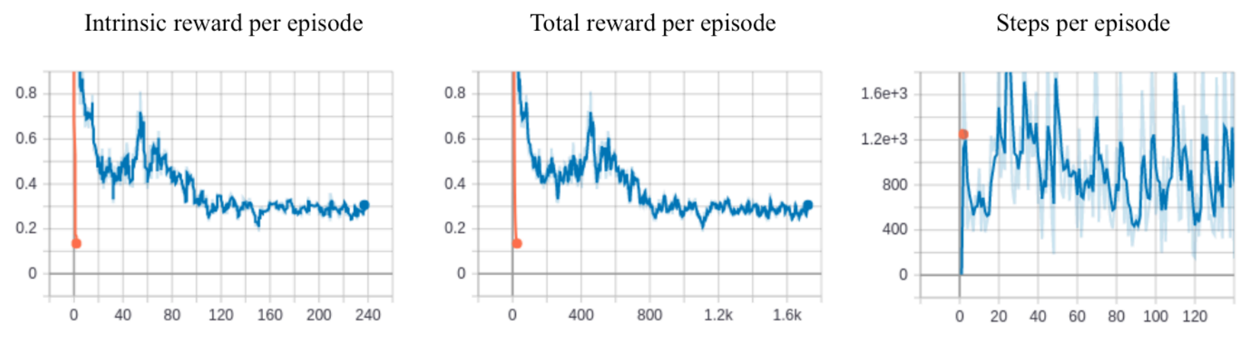

Figure 4.3.2: The result of playing Super Mario Bros game in RND algorithm. First two graphs show the reward per episode and total reward for 250 episode, and the last graph shows the number of action that agent takes per episode.

4.3.2 Mario Using RND Method

We then let the agent learn the game using RND algorithm. This time agent is able to move forward smoothly and manage to complete Stage 1 in 250 episodes. The results of intrinsic reward and steps number are shown in Figure 4.3.2. We can observe that the general trends of both intrinsic reward and step number are coherent with that of previous games. The intrinsic reward roughly decrease as episode number increase. On the other hand, third graph shows that step number vibrates among 1000 no matter how episode number changes. These results are very close to what we observe in previous games.

By contrast, the performance of RND is much better than the performance than ICM. The first graph shows the intrinsic reward generated by RND throughout the training process. We can see that agent continuously explores new environment. After training for 250 episodes, RND is able to complete the first level without dying and advance to half of second level, which is very impressive compared to the progress of ICM. The third graph indicates that RND method lives longer than ICM as well. Contrast to ICM that agent usually sticks at the starting position, the step number of RND vibrates mainly between 400 to 1800 which is much more natural and behaves more like human player. Moreover, we observe that agent interacting with other items in game environment during moving. We see that agent not only manages to hit bricks to collect coins and mushrooms, but also steps on the top of enemies to kill them. The greatest surprise of all is that agent has entered a hidden room by squatting down on the top of the green pipe in first level. Although we could not arrive at a conclusion that agent performs those actions intentionally since it does not try to collect all the collectable items during journey, we still think that this result is satisfactory enough and believe that RND could achieve better performance if we continue the training process.

In conclusion, the results show that RND performs better than ICM in both output score and training time in the later two game environments. We consider the reason might be that both Montezuma and Super Mario have complex environments which require agent to acquire ability to explore novel environment intensely. RND undoubtedly meets this requirement better than ICM, which indicates the positive advantage of adding the predictor network on the top of vanilla ICM structure. In next section, we will present further discussion about some challenges we face in curiosity-driven agent and how we might improve the performance.

CHAPTER 5

Discussion

In this section, we will present some challenges we face of using curiosity as intrinsic motivation as well as discuss some ideas that might potentially improve the performance of curiosity-driven agent.

5.1 Challenges in Curiosity-based method

As introduced previously, human, as well as other living beings, first observes the environment, then gets familiar with it, and finally take actions to achieve certain goal. Real life situations are undoubtedly more complex than game environments. Therefore agent requires some kind of motivation, such as curiosity, to firstly explore and learn the environment before they discover the method to solve the problems.

However, not all the tasks can be solved by curiosity alone. In addition to curiosity, we still need to set a goal for agent. Without a specific, or say long-term goal, stuffs that agent explored driven by its intrinsic curiosity could just be useless and meaningless rewards. Agents might be able to learn and explore everything that seems to be novel or unusual to them, but those learned knowledge can barely help them solve the problems. In fact, agent might not need to learn everything in the environment. It might only requires a subset which shall be sufficient for agent to achieve the goal.

For most of the time, agents have to determine which is more important between exploring the environment and accomplishing the task. For instance the Super Mario experiment, agent driven by the curiosity is able to finish the first level through simply moving forward. However, if we consider the Noisy-TV problem in first level, we are not sure whether agent would stop in front of and start to watch TV instead of keep going forward, according to its intrinsic curiosity. It is hard to determine the importance between exploring more states to fulfill its curiosity and capturing the ultimate goal. The tradeoff between these two need further consideration.

Moreover, real life situations are more complex and unpredictable than game environments where the environment and ultimate goal in real life are always not being defined clearly. In real life situations, states are more unpredictable and environment could always be changing so that the exploration done before might be useless.

In addition, the performance of curiosity driven method remains unknown in multi-agents scenario where agents could interact with each other. Agents driven by curiosity could be curious about other agents and might prefer to exchange information with others to accelerate their exploring process. The competition and collaboration between curiosity-driven agents remains an interesting topic for future researchers.

5.2 Introducing Human Prior

One possible idea of improvement we can try is to incorporate some human prior as suggested by [8]. Reinforcement learning algorithms do not understand the semantic meaning of the pixels on the screen, and need to learn everything from scratch. Humans start the game with a lot of prior knowledge of the game. For example, the ladder can be climbed to travel vertically, or the snake should often be avoided. This kind of knowledge allow humans to quickly achieve great results in the games.

Another advantage of incorporating prior knowledge is that agent can acquire stronger ability in imitating human behavior. Agent can display more efficient actions and the movements of agent could be more purposeful as well. In addition, human could also control the agent's learning preference by having human intentionally provides certain prior knowledge so that agent is able to deal with complex tasks more handy.

5.3 Other internal motivations

In addition to curiosity, there are other intrinsic motivation that could used in RL as well. One motivation we discuss here is hindsight. Hindsight is not a new idea in RL problem. There are many back propagation methods take use of the idea of hindsight to learn from error. For us human, the idea of hindsight also appears in our everyday life. Unlike curiosity that is often used to get familiar with the environment, in real life situation we often feel regret doing something wrong and start to blame ourself to learn from the error. Then in next time we will know that what should be the correct way to solve the problem. This idea of hindsight could be incorporated with curiosity for agent's learning process.

In RL game environment, agent with the idea of hindsight could realize that certain actions will result in losing its life. Then after learning from the error, agent will try to explore the environment to figure out the correct actions that can help itself achieve higher score. Combining curiosity with hindsight not only enables agent to explore new states efficiently but also courages agent to discover correct actions to reach the final goal.

CHAPTER 6

Conclusion

In this work, we have seen how curiosity could affect the performance of agents in sparse reward environments. Curiosity is an important factor that motivates and gives agents an intrinsic reward to explore and get familiar with the environment. Especially in those situations where the extrinsic rewards are absent or long-term, curiosity gives the agents short-term immediate rewards throughout the progress that help them build the concepts of the surrounding environment. During the experiment, we have observed that how Random Network Distillation could influence the results in a positive way when compared to the vanilla Intrinsic Curiosity Module method. However, curiosity itself alone might not be the only intrinsic motivation for human, in future work we will try to combine other internal rewards with curiosity to further improve the performance.

BIBLIOGRAPHY

[1] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. *Human-level control through deep reinforcement learning*. Nature, 2015.

[2] Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. *Continuous control with deep reinforcement learning*. ICLR, 2016.

[3] Ryan, Richard; Deci, Edward L. *Intrinsic and extrinsic motivations: Classic definitions and new directions.* Contemporary Educational Psychology, 2000.

[4] Silvia, Paul J. *Curiosity and motivation*. In The Oxford Handbook of Human Motivation, 2012.

[5] Oudeyer, Pierre-Yves and Kaplan, Frederic. *What is intrinsic motivation? a typology of computational approaches.* Frontiers in neurorobotics, 2009.

[6] Oudeyer, Pierre-Yves, Kaplan, Frdric, and Hafner, Verena V. *Intrinsic motivation systems for autonomous mental development.* Evolutionary Computation, 2007.

[7] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. *Curiosity-driven exploration by self-supervised prediction.* In International Conference on Machine Learning (ICML), 2017.

[8] R. Dubey, P. Agrawal, D. Pathak, T. L. Griffiths,and A. A. Efros. *Investigating human priors forplaying video games.* In ICML, 2018.

[9] Sun, Yi, Gomez, Faustino, and Schmidhuber, Jurgen. *Planning to be surprised: Optimal bayesian exploration in dynamic environments*. In AGI, 2011.

[10] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. *Diversity is all you need: Learning diverse skills without a reward function*. 2018.

[11] Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. *Unifying count-based exploration and intrinsic motivation*. In NIPS, 2016.

[12] Lopes, Manuel, Lang, Tobias, Toussaint, Marc, and Oudeyer, Pierre-Yves. *Exploration in model-based reinforcement learning by empirically estimating learning progress*. In NIPS, 2012.

[13] Houthooft, Rein, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. Vime. *Variational information maximizing exploration.* In NIPS, 2016.

[14] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. *Exploration by random network distillation.* 2018.

[15] T. Simonini. *Random Network Distillation: a new take on Curiosity-Driven Learning.* In data from the trenches, 2019.