Exploration Strategies in Deep Reinforcement Learning

1st Rakesh Reddy Kondeti
Robotics and Autonomous Systems
Universität zu Lübeck
Lübeck, Germany
rakesh.kondeti@student.uni-luebeck.de

2nd Honghu Xue
Robotics and Autonomous Systems
Universität zu Lübeck
Lübeck, Germany
xue@rob.uni-luebeck.de

***Abstract*—The fundamental issue in reinforcement learning is the balance between exploration of the environment and exploitation of the information already obtained by the agent.This paper surveys several exploration strategies used in reinforcement learning and summarizes the existing research with respect to their applicability and effectiveness. Starting off with the classic exploratory strategies, the paper addresses Intrinsic Rewards as Exploration Strategies, Memory-based exploration and Variational approaches. This paper investigates the performances/limitation of each approach and challenges in this field.**

***Index Terms*—Reinforcement Learning, Exploration strategies**

I. Introduction

Reinforce Learning (RL) has becoming more and more prevalent in real world. Loosely defined, reinforcement learning is a branch of artificial intelligence, in which an agent learns to control (or behave in) an unknown environment by interacting with that environment. Typically, the agent's performance is evaluated based on "understanding" of its environment. Clearly, an important factor of this problem has to do with how the agent learns about the process of discovering its environment(exploration).

One of the most successful applications of reinforcement learning is Gaming development. The application in other domains like robotics, self-driving cars, deep learning is already proving to be successful. The implementation of model-free reinforcement learning into Atari games is outperforming humans by making better decisions. Yet, unlike the vast majority of the Atari games, which are now easily solved at superhuman level by learned agents, Montezuma's Revenge has been hitherto unsolved by Deep Reinforcement Learning methods and was thought by some to be unsolvable for years to come. One of the reasons for this could be the current available exploration strategies may not be able to explore the Montezuma's Revenge properly. Hence, exploration strategies are very much important to explore the environment properly, which inturn helps the agent in making better decisions.

This report surveys the common exploration strategies employed in reinforcement learning and summarizes their effectiveness and applicability. There exists a large body of research pertaining to exploration in reinforcement learning, and more generally, to the problem of automated decision making in the presence of uncertainty. This report does not attempt to summarize or distil that wealth of information into a unified document. Rather, this report serves as an introductory overview of the broad classes and categories of approaches that are documented in the available literature.

The report begins with a brief introduction to the role of exploration in reinforcement learning. In particular, the trade-off between exploring the environment and exploiting that which the agent has already learned is discussed. This is followed by the overview of the commonly employed strategies and techniques. Finally, a summary of the merits and comparative effectiveness of each strategy is provided.

II. Background Knowledge

The goal of the Reinforcement Learning problem is to maximize the expected cumulative rewards, there by allowing the agent to learn the optimal policy. The formula is shown as,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \Sigma_{k=0}^{\infty} \gamma^k R_{t+k+1}. \quad (1)$$

A *reward* R_t is a scalar feedback signal that indicates how well an agent is performing at time point t. The *return* G_t is the expected total rewards from time step t into the future, The parameter $\gamma \in [0, 1)$ is a *discount factor* to avoid infinite returns. When $\gamma = 0$, the total return G_t only depends on immediate reward R_{t+1}.

A. The Role of Exploration

Exploration is an attempt to maximize knowledge gain and is one of the first steps that a reinforcement learning agent would take. Exploration strategy directly influences parameters like learning time, and speed and cost of learning of an agent. The more effectively and efficiently an agent explores and learns from its environment, the more effectively and efficiently it uses its time, and hence, the less time is required for learning.

However, pure exploration during learning would not be the most effective learning approach as it may waste much time and computing resources exploring task-irrelevant parts of the environment. On contrary, the smaller the learning time, the larger the cost (i.e., the poorer the performance of

the agent during its learning phase), and vice versa. So, it is often advantageous to find some suitable balance of both exploration and exploitation. If the agent can exploit its current knowledge of the environment, it may be able to identify the most worthwhile parts of the environment to explore.

Exploitation versus exploration is a critical topic in Reinforcement Learning. We would like the RL agent to find the best solution as fast as possible. However, in the meantime, committing to solutions too quickly without enough exploration does not yield optimal solution, as it could lead to local minima or total failure. Modern RL algorithms that optimize for the best returns can achieve good exploitation quite efficiently, while exploration remains more like an open topic

B. Classical Exploration Strategies

As a quick recap, this section covers several classical exploration algorithms that work out pretty well to solve simple RL problem

Epsilon-greedy: The agent does random exploration occasionally with probability and takes the optimal action most of the time with probability $1 - \epsilon$.

Upper confidence bounds: The agent selects the greediest action to maximize the upper confidence bound $Q_t(a)+U_t(a)$, where $Q_t(a)$ is the average rewards associated with action a up to time t and $U_t(a)$ is a function reversely proportional to how many times action a has been taken. See here for more details.

Boltzmann exploration: The agent draws actions from a Boltzmann distribution (softmax) over the learned Q values, regulated by a temperature parameter τ.

Thompson sampling: The agent keeps track of a belief over the probability of optimal actions and samples from this distribution

III. Exploration Strategies

A. Intrinsic Rewards as Exploration Bonuses

The current problems with extrinsic rewards is that this function is coded by human, which is not scalable to the real world problems(such as designing a good reward function for autonomous vehicles).

One common approach to better exploration, especially for solving the hard-exploration problem, is to augment the environment reward with an additional bonus signal to encourage extra exploration. The policy is thus trained with a reward composed of two terms, $r_t = r_t^e + \beta r_t^i$, where β is the hyper-parameter adjusting the balance between exploitation and exploration [1].

- r_t^e is an extrinsic reward function from the environment at time t
- r_t^i is an intrinsic exploration bonus at time t.

The main idea behind intrinsic rewards is that the exploratory strategies should be rewarded intrinsically just like intrinsic rewards obtained in the human mind like surprise, familiarity, curiosity,etc,. In reinforcement learning, these rewards are roughly categorised into two categories: 1) Discovery of novel states and 2) Improvement of the agent's knowledge about the environment.

1) Count Based Exploration

To know whether a state is a surprise or familiar, we need a way to measure whether a state is novel or appears often. One intuitive way is to count how many times a state has been encountered and to assign a bonus accordingly. The bonus guides the agent's behavior to prefer rarely visited states to common states. This is known as the *count-based exploration* method.

Let $N_n(s)$ be the *empirical count* function that tracks the real number of visits of a state **s** in the sequence of $s_{1:n}$. unfortunately, using $N_n(s)$ is not practical as most of the state would have $N_n(s) = 0$, as state space is mostly continuous or high-dimensional. We need an non-zero count for most states, even when they haven't been seen before [1].

Counting by Density Model:

density model to approximate the frequency of state visits and a novel algorithm for deriving a pseudo-count from this density model [3]. The conditional probability over the state space is defined as , $\rho_n(s) = \rho(s|s_{1:n})$, as the probability of the *(n+1)-th* state being *s* given the first *n* states are $s_{1:n}$. To measure this empirically, we can simply use $N_n(s)/n$ [1].

The *recoding probability* of a state s is defined as the probability assigned by the density model to s after observing a new occurrence of *s*, $\rho'_n(s) = \rho(s|s_{1:n}s)$,

$$\rho_n(s) = \frac{\hat{N}_n(s)}{\hat{n}} \leq \rho'_n(s) = \frac{\hat{N}_n(s)+1}{\hat{n}+1}. \tag{2}$$

The relationship between $\rho_n(s)$ and $\rho'_n(s)$ requires the density model to be learning-positive. In other words, After observing one instance of **s**, the density model's prediction of that same s should increase. Apart from being learning-positive, the density model should be trained completely online with non-randomized mini-batches of experienced states, so naturally we have $\rho'_n = \rho_{n+1}$.

By solving the the above linear equation,

$$\hat{N}_n(s) = \hat{n}\rho_n(s) = \frac{\rho_n(s)(1-\rho'_n(s))}{\rho'_n(s)-\rho_n(s)}. \tag{3}$$

The most common choice of intrinsic bonus is $r_t^i = N(s_t, a_t)^{(-1/2)}$ [4]. The pseudo-count-based exploration bonus is shaped in a similar form $r_t^i = (\hat{N}(s_t, a_t)+0.01)^{(-1/2)}$ [1]

Counting after Hashing:

Another idea to make it possible to count high-dimensional states is to map states into *hash codes* so that the occurrences of states become trackable [2].The main idea is that, the sate space is discretized with a hash function $\phi : \mathbb{S} \to \mathbb{Z}^{\mathbb{K}}$. An exploration bonus r^i: $\mathbb{S} \to \mathbb{R}$ is added to the reward function,

defined as $r^i(s) = N(\phi(s))^{(-1/2)}$, $N(\phi(s))$ is an empirical count of occurrences of $\phi(s)$.

Locality-Sensitive Hashing (LSH) is proposed to convert continuous, high-dimensional data to discrete hash codes [2]. LSH is a popular class of hash functions for querying nearest neighbors based on certain similarity metrics. A hashing scheme $x \rightarrow h(x)$ is locality-sensitive if it preserves the distancing information between data points, such that close vectors obtain similar hashes while distant vectors have very different ones.

2) Prediction based Exploration

The main idea of Count-based approaches is to keep a count on previously visited states, and give bigger rewards to novel states. The disadvantage of this approach is that it tends to become less effective as the number of possible states grows.

The second category of intrinsic exploration bonuses are rewarded for improvement of the agent's knowledge about the environment. The agent's familiarity with the environment dynamics can be estimated through a prediction model. This idea of using a prediction model to measure curiosity was actually proposed quite a long time ago [5].

Forward Dynamics:

Learning a *forward dynamics prediction model* is a great way to approximate how much knowledge our model has obtained about the environment and the task MDPs. It captures an agent's capability of predicting the consequence of its own behavior, f:$(s_t, a_t) \rightarrow s_{t+1}$. Such a model cannot be perfect, the error $e(s_t, a_t)$ = $||f(s_t, a_t) - s_{t+1}||_2^2$ can be used for providing intrinsic exploration rewards. The higher the prediction error, the less familiar we are with that state. The faster the error rate drops, the more learning progress signals we acquire [1].

Intelligent Adaptive Curiosity (IAC) sketched an idea of using a forward dynamics prediction model f to estimate learning progress and assigned intrinsic exploration reward accordingly [6]. IAC relies on a memory which stores all the experiences encountered by the robot, $M = \{(s_t, a_t, s_{t+1})\}$ and a forward dynamics model f. IAC incrementally splits the state space (i.e. sensorimotor space in the context of robotics) into separate regions based on the transition samples, using a process similar to how a decision tree is split: The split happens when the number of samples is larger than a threshold, and the variance of states in each leaf should be minimal. Each tree node is characterized by its exclusive set of samples and has its own forward dynamics predictor f, named "expert". The prediction error e_t of an expert is pushed into a list associated with each region. The learning progress is then measured as the difference between the mean error rate of a moving window with offset τ and the current moving window. The intrinsic reward is defined for tracking the learning progress: $r_t^i = \frac{1}{k}\sum_{i=0}^{k-1}(e_{t-i-\tau} - e_{t-i})$ where k is the moving window size. So the larger prediction error rate decrease we can achieve, the higher intrinsic reward we would assign to the agent. In other words, the agent is encouraged to take actions to quickly learn about the environment.

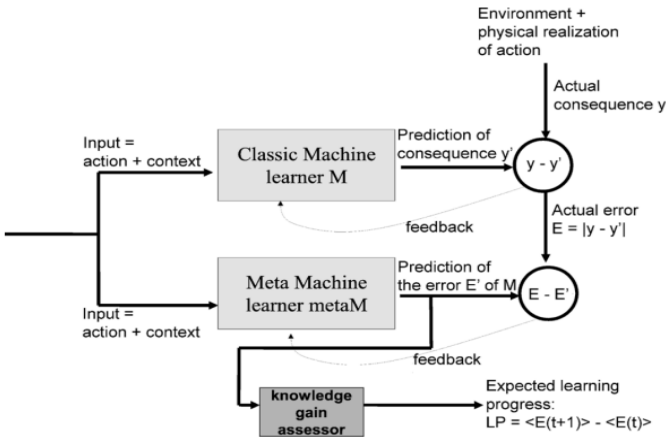

Fig. 1. Architecture of IAC(Intelligent Adaptive Curiosity)(Image source: [6])

A Forward dynamics model is trained in the encoding space defined by ϕ, $f_\phi : (\phi(s_t), a_t) \rightarrow \phi(s_{t+1})$. The model's prediction error at time T is normalized by the maximum error up to time t, $\bar{e}_t = \frac{e_t}{max_{i \leq t} e_i}$, so it is always between 0 and 1. The intrinsic reward is defined as $r_t^i = \left(\frac{\bar{e}_t(s_t, a_t)}{t.C}\right)$, where $C > 0$ is a decay constant [7].

Encoding the state space via $(.)$ is necessary, as the experiments have shown that, a dynamics model trained directly on raw pixels has very poor behavior — assigning same exploration bonuses to all the states. The encoding function ϕ is learned via an autocoder (AE) and $\phi(.)$ is one of the output layers in AE. The AE can be statically trained using a set of images collected by a random agent, or dynamically trained together with the policy where the early frames are gathered using -greedy exploration [7].

Intrinsic Curiosity Module learns the state space encoding $\phi(.)$ with a self-supervised *inverse dynamics* model. Predicting the next state given the agent's own action is not easy, especially considering that some factors in the environment cannot be controlled by the agent or do not affect the agent. ICM believes that a good state feature space should exclude such factors because they cannot influence the agent's behavior and thus the agent has no incentive for learning them. By learning inverse dynamics model $g : (\phi(s_t), \phi(s_{t+1})) \rightarrow a_t$, the feature space only captures those changes in the environment related to the actions of our agent, and ignores the rest [8].

Given a forward model f, an inverse dynamics model g and an observation (s_t, a_t, s_{t+1}):

$$g_{\psi_I}(\phi(s_t), \phi(s_{t+1})) = \hat{a}_t; f_{\psi_F}(\phi(s_t), a_t) = \hat{\phi}(s_{t+1})$$

$$r_t^i = ||\hat{\phi}|(s_{t+1}) - \phi(s_{t+1})||_2^2 \quad (4)$$

Such $\phi(.)$ is expected to be robust to uncontrollable aspects of the environment.

A set of large scale experiments were performed purely based on curiosity-driven learning (only intrinsic rewards are

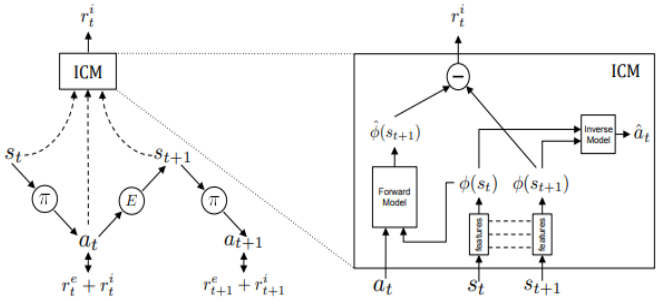

Fig. 2. Intrinsic Curiosity Module (Image source: [8])

provided to agent) [9].A good choice of ϕ is crucial to learning forward dynamics, which is expected to be *compact*, *sufficient* and *stable*, making the prediction task more tractable and filtering out irrelevant observation.

Some of the encoding functions:

- Raw image pixels: No encoding, $\phi(x) = x$
- Random features (RF): Each state is compressed through a fixed random neural network
- The probabilistic encoder is used for encoding, $\phi(x)$=$q(z|x)$
- Inverse dynamic features (IDF): The feature space as used in ICM

Random Networks:

But, what if the prediction task is not about the environment dynamics at all? It turns out when the prediction is for a random task, it still can help exploration.

DORA(Directed Outreaching Reinforcement Action-Selection) is a novel framework that injects exploration signals based on a newly introduced, task-independent MDP [10]. The two parallel MDPs used in DORA:

- Original task MDP
- Other identical MDP but with no reward attached. Rather, every state-action pair is designed to have value 0. The Q-value learned for the second MDP is called *E-value*. If the model cannot perfectly predict E-value to be zero, it is still missing information.

Initially E-value is assigned with value 1. Such positive initialization can encourage directed exploration for better E-value prediction. State-action pairs with high E-value estimation don't have enough information gathered yet. To some extent, the logarithm of E-values can be considered as a generalization of visit counters.

Similar to DORA, Random Network Distillation [11] introduces a prediction task independent of the main task. The RND exploration bonus is defined as the error of a neural network $\hat{f}(s_t)$ predicting features of the observations given by a fixed randomly initialized neural network $f(s_t)$. The motivation is that given a new state, if similar states have been visited many times in the past, the prediction should be easier and thus has lower error. The exploration bonus is $r(s_t^i)$ $= ||\hat{f}(s_t;\theta) - f(s_t)||_2^2$

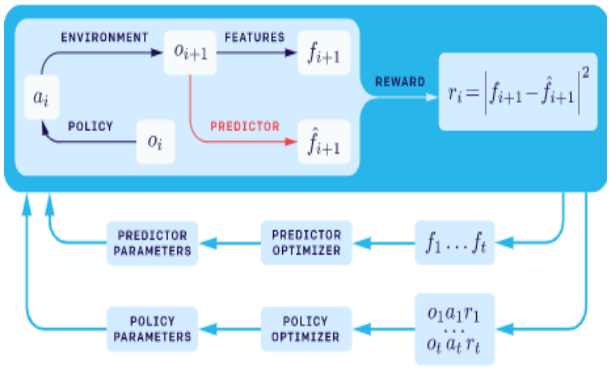

Fig. 3. Random Network Distillations (Image soure: [1])

Factors important in RND experiments:

- Non-episodic setting results in better exploration, especially when not using any extrinsic rewards. It means that the return is not truncated at "Game over" and intrinsic return can spread across multiple episodes.
- Normalization is important since the scale of the reward is tricky to adjust given a random neural network as a prediction target. The intrinsic reward is normalized by division by a running estimate of the standard deviations of the intrinsic return.

Deep RL in Robotic Applications:

Different from games in simulators, some RL applications like Robotics need to understand objects and intuitive reasoning in the physical world. Some prediction tasks require the agent to perform a sequence of interactions with the environment and to observe the corresponding consequences, such as estimating some hidden properties in physics (e.g. mass, friction, etc).

DRL agents can learn to perform necessary exploration to discover such hidden properties [12]. Precisely they considered two experiments:

- *"Which is heavier?"* - Agent interacts with the blocks and infer which one is heavier.
- *"Towers"* - The agent needs to infer how many rigid bodies a tower is composed of by knocking it down

The agent in the experiments first goes through an exploration phase to interact with the environment and to collect information. Once the exploration phase ends, the agent is asked to output a labeling action to answer the question. Then a positive reward is assigned to the agent if the answer is correct; otherwise a negative one is assigned. Because the answer requires a decent amount of interactions with items in the scene, the agent has to learn to efficiently play around so as to figure out the physics and the correct answer.

B. Memory-based Exploration

1) Episodic Memory

Random Network Distillation is better running in an non-episodic setting, meaning the prediction knowledge is accumulated across multiple episodes. The exploration strategy,

Never Give Up [13], combines an episodic novelty module that can rapidly adapt within one episode with RND as a lifelong novelty module.

Precisely, the intrinsic reward in NGU consists of two exploration bonuses from two modules, within one episode and across multiple episodes, respectively.

The short-term per-episode reward is provided by an *episodic novelty module*. It contains an episodic memory **M**, a dynamically-sized slot-based memory, and an IDF (inverse dynamics features) embedding function ϕ,

- At every time step, the current state embedding $\phi(s_t)$ is added into **M**
- The intrinsic bonus is determined by comparing how similar the current observation is to the content of **M**.

$$r_t^{episodic} \approx \frac{1}{\sqrt{\sum_{\phi_i \in N_k} K(\phi(x_t), \phi_i)} + c} \tag{5}$$

where $K(x, y)$ is a kernel function for measuring the distance between two samples. N_k is a set of k nearest neighbours in M. c is a small constant to keep the denominator non-zero. In the paper [13] $K(x, y)$ is configured as $K(x, y) = \frac{\epsilon}{\frac{d^2(x,y)}{d_m^2} + \epsilon}$, where $d(.,.)$ is Euclidean distance between two samples and d_m is the running average of squared Euclidean distance of the k-th nearest neighbours for better robustness, ϵ is a small constant.

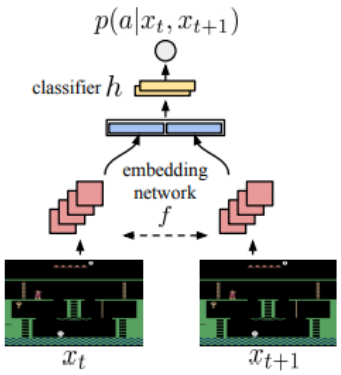

Fig. 4. NGU's training architecture (Image source: [13])

The long-term across-episode novelty relies on RND prediction error in $life-long\ novelty\ module$. The exploration bounus is $_t = 1 + \frac{e^{RND}(s_t) - \mu_e}{\sigma_e}$, where μ_e and σ_e are the running mean and standard deviation for RND error $e^{RND}(s_t)$.

The final combined intrinsic reward is $r_t^i = r_t^{episodic}.\text{clip}(\alpha_t, 1, L)$, where L is a constant maximum reward scalar.

The design of NGU enables two properties:

- Rapidly discourages revisiting the same state within the same episode;
- Slowly discourages revisiting states that have been visited many times across the episodes.

Instead of using Euclidean distances to measure the closeness of states, the paper [14] proposed to take the transition

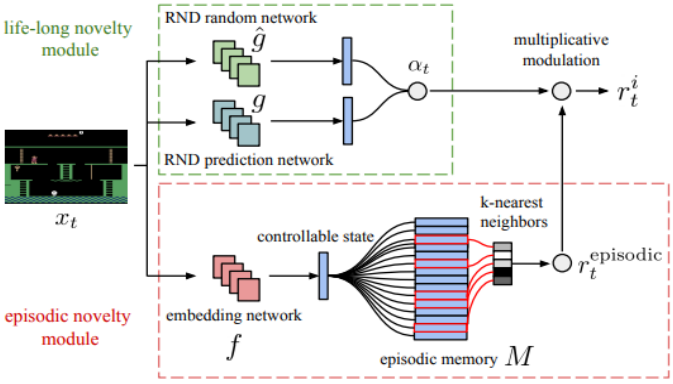

Fig. 5. NGU's reward genertor (Image source: [13])

between states into consideration and to take number of steps needed to visit one state from another in memory, name **Episodic Curiosity(EC)** module. The novelty bonus depends on reachability between states.

- At the beginning of each episode, the agent starts with an empty episodic memory M.
- At every step, the agent compares the current state with saved states in memory to determine novelty bonus: If the current state is novel (i.e., takes more steps to reach from observations in memory than a threshold), the agent gets a bonus.
- The current state is added into the episodic memory if the novelty bonus is high enough.

C. Direct Exploration

Go-Explore is an algorithm aiming to solve the "hard-exploration" problem. It is composed of the following two phases [15]. $First\ Phase\ Explore$: This stage does not use any neural nets or other machine learning algorithms but is solely based on random (maybe semi-guided) exploration. Here the authors introduced the idea of cells. Cells are downscaled and grayscale images of the actual game frames

Fig. 6. Original Image

The main goal here is to find interesting cells. Interesting cells are those that have not been found before and where the agent collected a high reward to get there. If a cell is encountered that is already in the archive the corresponding entry is updated in case it is 'better', i.e. higher total reward or shorter trajectory. This first phase made a huge implicit assumption: that our environment is deterministic. This is easy to achieve in a computer game, especially in a simple Atari

Fig. 7. Downscale Image(cell)

game, because here the same actions will always lead to the same results. Thus, second phase ("Robustification") is needed to robustify the solution via imitation learning. They adopted Backward Algorithm, in which the agent is started near the last state in the trajectory and then runs RL optimization from there.

However, to make the algorithm more generally useful to environments with stochasticity, an enhanced version of Go-Explore, named policy-based Go-Explore was proposed later.

D. Varitional Options

Options are policies with termination conditions. There are a large set of options available in the search space and they are independent of an agent's intentions. By explicitly including intrinsic options into modeling, the agent can obtain intrinsic rewards for exploration.

Variational Intrinsic Control [16] is such a framework for providing the agent with intrinsic exploration bonuses based on modeling options and learning policies conditioned on options. Let Ω represent an option which starts from $s0$ and ends at s_f. An environment probability distribution $p^J(s_f|s_0, \Omega)$ defines where an option Ω terminates given a starting state s_0. A controllability distribution $p^C(\Omega|s_o)$. defines the probability distribution of options we can sample from.

$$p(s_f, \Omega|s_o) = p^J(s_f|s_0,)p^C(\Omega|s_0) \tag{6}$$

While choosing options, we would like to achieve two goals:

- Achieve a diverse set of the final states from $s_0 \implies$ Maximization of $H(s_f|s_0)$.
- Know precisely which state a given option Ω can end with $\implies$ Minimization of $H(s_f|s_0, \Omega)$

Different from VIC which models Ω conditioned only on start and end states, *VALOR*(*Variational Auto-encoding Learning of Options by Reinforcement*) [17] relies on the whole trajectory to extract the option context c, which is sampled from a fixed Gaussian distribution.

- A policy acts as an encoder, translating contexts from a noise distribution into trajectories
- A decoder attempts to recover the contexts from the trajectories, and rewards the policies for making contexts easier to distinguish. The decoder never sees the actions during training, so the agent has to interact with the environment in a way that facilitates communication with the decoder for better prediction.

IV. CONCLUSION

Reinforcement Learning depends upon reward hypothesis and return maximization, which is the idea that each goal can be described as the maximization of the rewards. A good exploration strategy and the trade-off between exploration and exploitation is required to maximize the rewards. However, the current problems with extrinsic rewards is that this function is coded by human, which is not scalable to the real world problems(such as designing a good reward function for autonomous vehicles). Often the agent is also stuck in the local optima in case of hard-exploration and noisy TV problems. So, the basic idea of intrinsic rewards is proposed to solve these problems.

This paper discusses the importance of intrinsic exploration bonus strategies in RL agents. A strong emphasis is made on intrinsic rewards as the exploration bonus: Count based and Prediction based explorations. However, the basic overview of Memory based exploration and other state-of-the-art exploration strategies like Never Give Up algorithm and Variational options are presented. Finally, to give a better understanding, each exploration strategy is explained with respect to solving some task in gaming environment.

REFERENCES

[1] Weng, Lilian. "Exploration Strategies in Deep Reinforcement Learning." Journal - lilianweng.github.io/lil-log. 2020

[2] Tang, Haoran, et al. " exploration: A study of count-based exploration for deep reinforcement learning." 31st Conference on Neural Information Processing Systems (NIPS). Vol. 30. 2017.

[3] Bellemare, Marc G., et al. "Unifying count-based exploration and intrinsic motivation." arXiv preprint arXiv:1606.01868 (2016).

[4] Strehl, Alexander L., and Michael L. Littman. "An analysis of model-based interval estimation for Markov decision processes." Journal of Computer and System Sciences 74.8 (2008): 1309-1331.

[5] Schmidhuber, Jürgen. "A possibility for implementing curiosity and boredom in model-building neural controllers." Proc. of the international conference on simulation of adaptive behavior: From animals to animats. 1991.

[6] Oudeyer, Pierre-Yves, Frdric Kaplan, and Verena V. Hafner. "Intrinsic motivation systems for autonomous mental development." IEEE transactions on evolutionary computation 11.2 (2007): 265-286.

[7] Stadie, Bradly C., Sergey Levine, and Pieter Abbeel. "Incentivizing exploration in reinforcement learning with deep predictive models." arXiv preprint arXiv:1507.00814 (2015

[8] Pathak, Deepak, et al. "Curiosity-driven exploration by self-supervised prediction." International Conference on Machine Learning. PMLR, 2017.

[9] Burda, Yuri, et al. "Large-scale study of curiosity-driven learning." arXiv preprint arXiv:1808.04355 (2018).

[10] Choshen, Leshem, Lior Fox, and Yonatan Loewenstein. "Dora the explorer: Directed outreaching reinforcement action-selection." arXiv preprint arXiv:1804.04012 (2018).

[11] Burda, Yuri, et al. "Exploration by random network distillation." arXiv preprint arXiv:1810.12894 (2018).

[12] Denil, Misha, et al. "Learning to perform physics experiments via deep reinforcement learning." arXiv preprint arXiv:1611.01843 (2016).

[13] Badia, Adrià Puigdomènech, et al. "Never give up: Learning directed exploration strategies." arXiv preprint arXiv:2002.06038 (2020).

[14] Savinov, Nikolay, et al. "Episodic curiosity through reachability." arXiv preprint arXiv:1810.02274 (2018).

[15] Ecoffet, Adrien, et al. "Go-explore: a new approach for hard-exploration problems." arXiv preprint arXiv:1901.10995 (2019).

[16] Gregor, Karol, Danilo Jimenez Rezende, and Daan Wierstra. "Variational intrinsic control." arXiv preprint arXiv:1611.07507 (2016).

[17] Achiam, Joshua, et al. "Variational option discovery algorithms." arXiv preprint arXiv:1807.10299 (2018).