GO-EXPLORE: A NEW APPROACH FOR HARD-EXPLORATION PROBLEMS?

February 11, 2019

Ahmed BEN SAAD
Yanis DACI
Sajeevan PUVIKARAN

Contents

1 Introduction

The usual way to solve exploration problems is called Intrisic motivation. It consists on providing bonus rewards for novel states that encourage the agent to explore more of the state space even if it does not get external reward from the environment. The detail is in defining novelty and the scale of the intrinsic reward. The authors of the article explains that this method is not doing well as it call a phenomenon entitled detachment. They explain the concept with the following schema:

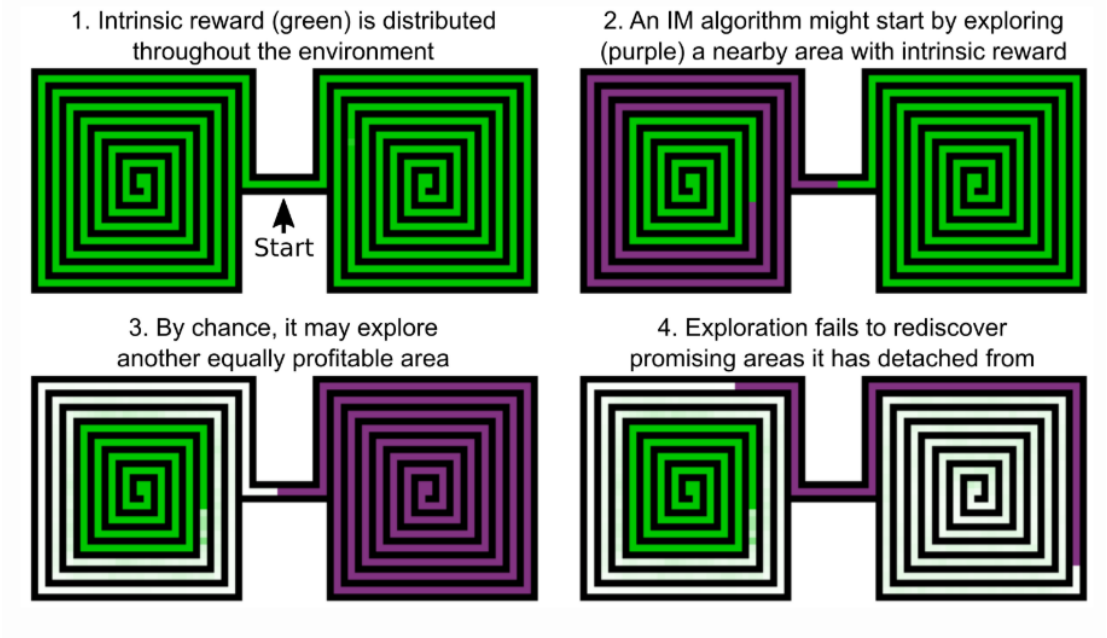


Figure 1: Example of detachment in intrinsic motivation (IM) algorithms.
Green areas indicate intrinsic reward, white indicates areas where no intrinsic reward remains, and purple areas indicate where the algorithm is currently exploring.

In this example, at the beginning, the agent is between two areas containing a lot of intrinsic rewards. Randomly, it only explores a part of the left spiral before exploring the right one. Consequently, a part of the left spiral will not be accessible because it is surrounded by states that are no longer novel.

Other limitations that one must consider in these problems where the agent needs to learn complex tasks with sparse/deceptive rewards like Montezuma's Revenge or Pitfall. For those examples, despite the huge research effort in RL, no algorithm made a score greater than 17,500 and no algorithm succeeded in scoring one single point in Pitfall which highlights the major problem in learning to interact with such complex environments.

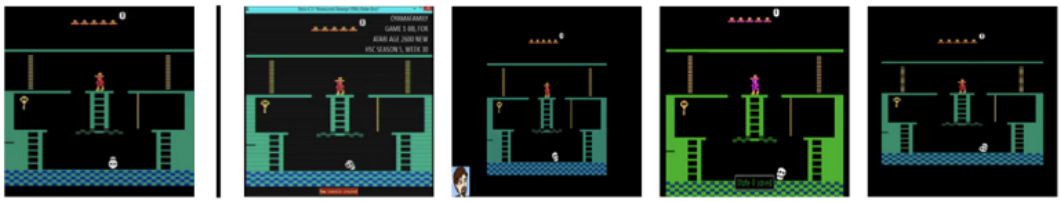
Thus, Uber proposed a solution: To maintain a memory of previously novel states. In this way, the agent will randomly try a previously visited state, in order to find new ones. They called this algorithm **Go-Explore**.

2 Related work

Solving this kind of hard exploration problems has been a subject of intense researches. Especially Montezuma's Revenge which has been seen as a kind of grand-challenge for Deep RL methods. In recent months, DeepMind and OpenAI claimed that they developed agents capable of solving the first levels of this game. They opted for Imitation Learning as an integral part of their algorithms. Imitation Learning is the procedure of learning to solve the game using human demonstrations which changes fundamentally the nature of the learning problem as it becomes more similar to a supervised learning technique rather than RL technique.

2.1 DeepMind

DeepMind used videos of experts solving the first levels of the game to support the learning process. The problem of learning from videos is in itself an interesting challenge, completely outside of the additional challenges posed by the game in question. As the authors point out, videos found on YouTube contain a various arrangement of artifacts which can prevent the easy mapping between what is happening in the video, and what an agent playing in the ALE might observe. In order to get around this "gap," they create a method which is able to embed observations of the game state (visual and auditory) into a common embedding space.



(a) ALE frame (b) Frames from different YouTube videos

Figure 2: Comparison of different demonstrations videos to emulator image.

2.2 OpenAI

A few weeks later, OpenAI shared a blog post claiming that they were able to train an agent to solve this game. Although it relies also on human demonstrations, they used them differently: Given an expert human trajectory through the game, the agent is started near the end of the game, and then slowly moved backwards through the trajectory on every restart as learning progresses. This has the effect of exposing the agent to only parts of the game which a human player has navigated through, and only widening the scope as the agent itself becomes more competent.

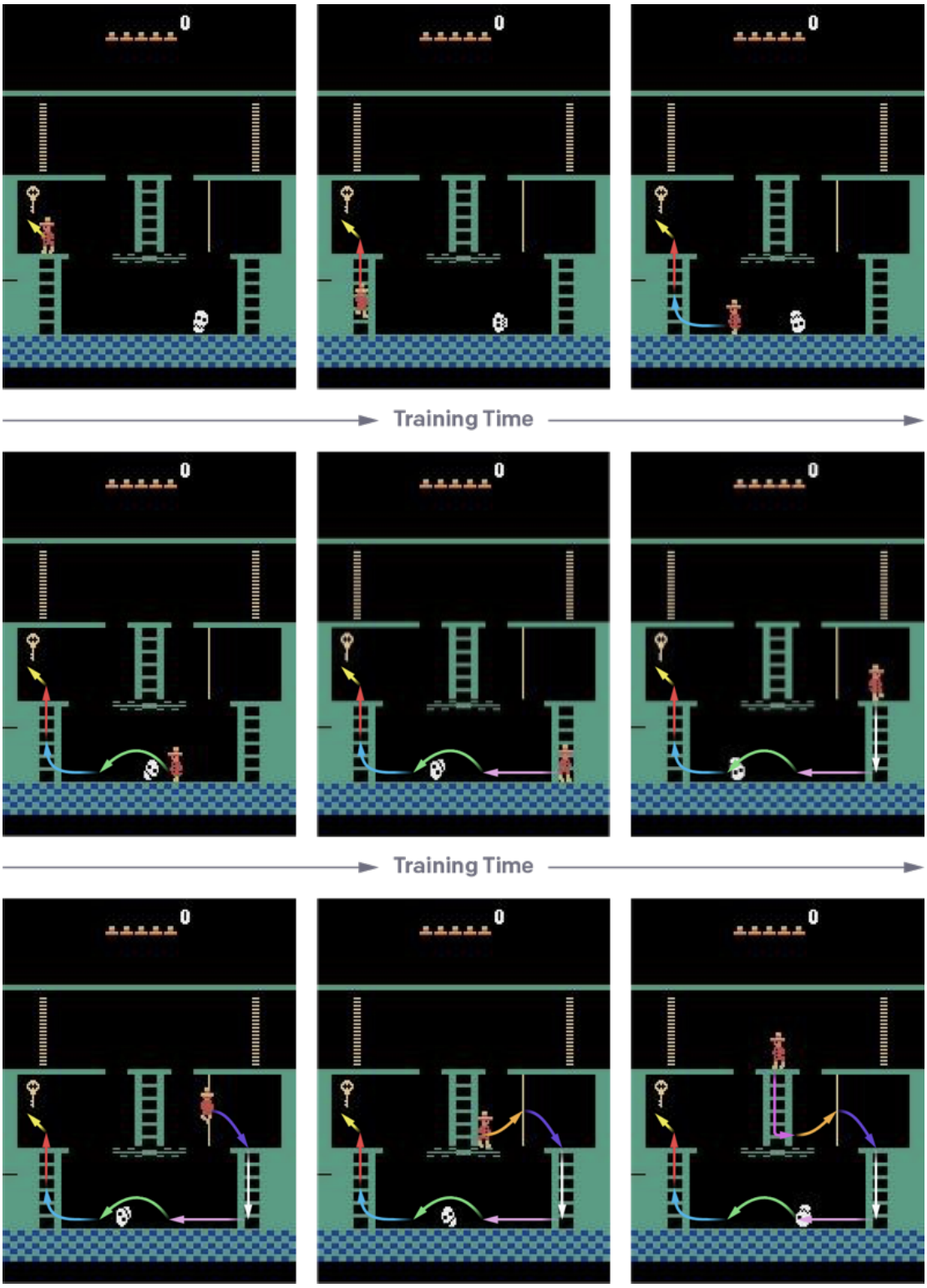


Figure 3: Restarts used during training over time.

Many other publications tried to solve similar problems (complex environment and sparse

rewards) but for the sake of simplicity we only focused on those two major results that used the same game as Go-Explore as environment and were considered as state of the art methods. So let's explore and discuss this new method to see what made it "explode" the score bar twice in Montezuma's Revenge.

3 Go-Explore

3.1 What is it?

Go Explore is a compound of two phases : **Explore until solved** and **robustify**.

Phase 1: Explore until solved

The Go Explore algorithm is based on this idea : During the learning phase, the agent randomly samples a previously visited state with the objective of finding newer ones. Consequently, the agent will travel to that state and will explore from this point. In the best-case scenario, it will find a new state near of the boundary of visited states and discover novel states not visited yet.

We can summarize this step with this algorithm :

Repeat until solved:

1. Choose a cell from the archive (optionally prefer newer cells)
2. Go back to that cell
3. Explore from that cell (randomly for n steps)
4. For all cells visited (including new cells), if the new trajectory is better (higher score), swap it in as the trajectory to reach that cell.

By storing a variety of stepping stones in an archive, Go-Explore remembers and returns to promising areas for exploration. Moreover, by first returning to cells before exploring from them, Go-Explore focuses on expanding its sphere of knowledge despite of over-exploring easily reached states.

Phase 2: Robustify (if necessary)

Finally, there is an optional robustification step. If the solutions found are not robust to noise, we can robustify them into a deep neural network with an imitation learning algorithm. Indeed, the trajectories learned from the algorithm can be sensitive to small deviations. Thus, to make the learned behavior more robust, we can proceed to imitation learning. In this case, we take the trajectories learned in a deterministic environment and we learn to reproduce them

in randomized versions of that environment.

We can summarize the Go-Explore algorithm with the following schema:

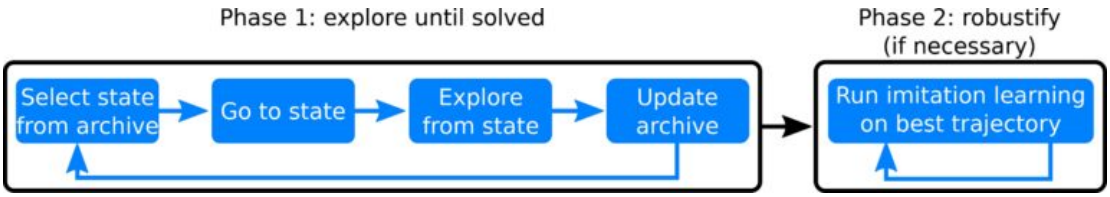
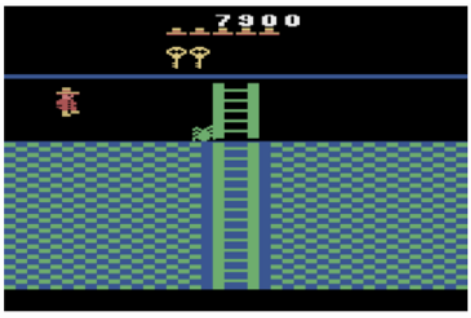


Figure 4: Go-Explore algorithm schema

3.2 The Techniques used

To execute the Go-Explore algorithm, they needed to find a way to **represent the game states within the memory**

Go-Explore needs a lower-dimensional cell representation which will be used to form its archive. This representation should conflate states that are similar enough to not be exploring separately. They found that simply downsampling the current game frame was sufficient to get a good enough code. The following figures show an example of downsampled cell representation:



Cell before downsampled cell representation

Cell after downsampled cell representation

Figure 5

Then, they had to find the best way to successfully **return to previously visited states**.

There are three ways to return to a cell, depending on the constraint of the environment. In order of efficiency, we have :

- In a **resettable environment**, we can reset the environment directly to that state.
- In a **deterministic environment**, we can replay the trajectory to the cell. It means that memorize and replay the sequence of action that take you to that state.

- In a **stochastic environment**, we can train a goal-conditioned policy that learns to reliably return to a cell

The first two methods make strong assumptions about the environment and are the only ones used in the reported results. Indeed, Atari is resettable. So, for efficiency reasons, they chose to return to previously visited cells by loading the game state. In the case of Montezuma's Revenge, it allowed to solve the first level 45 times faster than by replaying trajectories. Moreover, such access to the emulator is not required for Go-Explore to work. This fact makes it faster.

4 Experimental results

We decided to adapt a notebook found on a different game in order to improve our understanding and check the results of the article.

We worked on the the most popular Megadrive and SEGA game: Sonic The Hedgehog 2. For that purpose, we used Gym Retro. It is a wrapper for video game emulator cores using the Libretro API to turn them into Gym environments. It includes support for several classic game consoles and a dataset of different games.
Each game has files listing memory locations for in-game variables, reward functions based on those variables, episode end conditions, savestates at the beginning of levels and a file containing hashes of ROMs that work with these files.
We chose this game because the actions and the combination of valid moves which can be executed by Sonic are not very numerous. Moreover, we removed redundant and useless actions like look up.
Furthermore, when we created the Gym environment to execute our game, some very valuable information can be easily determined. Indeed, in the case of the Sonic games, the X axis and Y axis coordinates of the character, the level end bonus and the name of the stage played are available in the environment information. It is not the case for other games. Thus, we do not have to compute the location of Sonic which can be a very difficult task.

Now that we chose a game which can be studied, we had to adapt functions allowing to mimic the Go-Explore algorithm. We just worked on the "Explore until solved" phase. We took this decision because our algorithm would be too complex and difficult to put it into practice. Moreover, the "robustify" phase is not mandatory.

To achieve the exploration, the algorithm chooses at the first state a cell randomly. Then, it computes a multitude of trajectories with their new score and their associated actions. Afterwards, they are saved in an array to identify the best one. If this trajectory is better than the Sonic initial state which means that we reach a better score, we follow it. If the trajectory is not a better one, we randomly choose another cell and we repeat our computation of trajectories. We reiterate this method until we fulfill a winning condition. In our case, we consider that we reach our goal when the variable "level_end_bonus" is different from zero.

Finally, to visualize our results , we used a function to convert the chosen trajectories into a GIF. Therefore, we can proceed to a first evaluation of the algorithm.

Figure 6: GIF Representation of the trajectories chosen

We remarked that after 500 iterations which means approximately five minutes of training, we obtained a character jumping most of the time. It takes time to move forward in the level but it does not die which means that the results are not too bad. After one hour of training, the decisions taken are much better even if it can be improved. Indeed, when the algorithm has a higher training time, it can explore more areas and find better trajectories.
Here are the results that we found for the Sonic game:

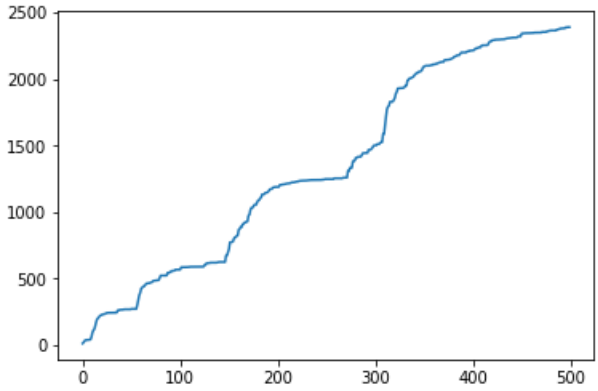


Figure 7: Number of cells covered

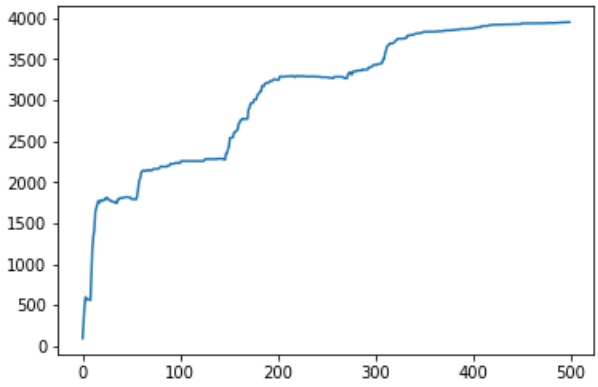


Figure 8: Number of trajectories found

We can remark that at the beginning of the exploration, we have a large number of new cells covered and trajectories found. Indeed, the more a character progress in the game, the less it will find new cells and trajectories. Moreover, we can observe "constant periods" on the figure 8. Indeed, the more we go through the iterations, the more these constant periods are time-consuming. We can explain this fact by a larger number of trajectories to verify.

5 Discussion

In a Reddit post -more than one month ago- one of the co-authors shared the blog post link and said that the official paper and the source code will be released soon. And obviously because of its nature, this publication doesn't present all the details about their results despite its research paper like structure. This approach had two main problems : First, some people asked about whether it was necessary to share "flashy results" without a paper nor source code so that the results can be checked. Second, and the most important, it rose many controversies. In this part we will focus on the latter problem.
One of the controversies that rose with Go-Explore is the way to represent game states that groups similar states and separate dissimilar ones. In the first part of the article they showed that downsampling the game frame was sufficient to achieve 35,000 points in Montezuma's Revenge which is already huge. But the agent that made 2,000,000 points and presented in their video used very specific domain knowledge (The position of the character, room number, number of keys, current level) which raises the question on whether their idea can be generalized to other environments and still achieve these super human scores. Again, the lack of source code (and sufficient details for implementation) made it hard for us to test Go-Explore on other games. But we will wait for the source code release and see. This may not be a big problem (a simple verification) and as the blog post presesnted two example in which Go-Explore outperformed other known RL techniques. But there are still major controversies: How do you successfully return to a previously visited state in the exploration phase ? Now this is a very important question and the article proposed 3 ways to deal with it :

- Reset the environment directly to the desired state by using save/load game frames. The problem with this idea is that it make the assumption that this is practically possible. It is wrong to say it can always go back to the cell, even in a deterministic setting. For example an autonomous car cannot go back to a previous state if it includes the fuel level. But it still works on simulation-based training phases which give it a good use case.

- Memorize and replay the sequence of actions that take you to that state. This requires a deterministic environment. Although the team said that working directly on stochastic environments by using a policy to return to previously visited states is a problem that we are hoping to tackle in future work.

- Learn a goal-conditioned policy. The authors stated that this method should work but was not tested. If it's true, this means that the stochastic-ness in training phase is no longer a problem as goal-conditionned policies can revisit already visited states in a stochastic environment. But if not, it will be a huge problem because it will mean that Go-Explore will be only suited for deterministic enivronment for training. Which is not very good as most of the RL real life problems are stochastic (e.g we will need simulators for training robots in real life, with all the limitations it can cause)

Although the last solution seem the best one, the results presented in the article are using the first 2 ideas.

We also believe that the exploration phase is a little bit problematic as the article says that they used random walks with 0.95 probability of repeating the previous actions. We think that this is only suited for Atari games-like problems where good policies include repeating the same actions many times (move forward in a room, attack an ennemy, etc...) which is clearly not the case for all RL problems. This raises the question of the ability of this explroation algorithm to be generalized for other use cases. One of the good things to look for in the future is a better and more intelligent action policy in this phase.
One should notice that the blog post recieved some critics saying that Go-Explore is not a RL algorithm saying that it is a planning algorithm that leverages strong heuristic knowledge (provided either by a human or a grid encoding that hardly seems generalisable), coupled with a *supervised* learning algorithm. Considering that the difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP. And again, saving and reloading states assumes that we have such knowledge. We need to wait for the research paper to verify all these claims.

6 Updates

After finishing this report and 3 days before the deadline, Uber Engineering labs published their research paper about Go-Explore and we found it on arXiv platform. It is a 37-page paper that developed some unclear points about their idea and answered some questions that were asked by the RL community. One of the main arguments given by the authors is that Go Explore allows agents to train on deterministic environments (e.g simulators) and then transfer the policy learned to a stochastic environment (e.g real world)

"The reliance on having access to a deterministic environment may initially seem like a drawback of Go-Explore, but we emphasize that deterministic environments are available in many popular RL domains, including videos games, robotic simulators, or even learned world models" But they also acknowledged the limits of the current exploration phase version and that this phase needs to be more "intelligent" instead of taking random walks. They proposed some solution as future work. For example they proposed to include more than one demonstration to the robustification phase to learn from. We think that this may be a good idea for future work to solve the problem of stochasticity because the multiple demonstrations provided to the second phase will take into account the stochasticness of the environment. But again, this is yet to be tried and we may need a huge number of demonstrations to achieve the same performances as in determinsitic training. In the same context and on the idea of returning to previous cells and how to do it, the authors stated that the environment training requires to be deterministic but they acknowledged that it was a source of controversy during the first annoucement of Go-Explore :

"Whether performing such resets is allowable for RL research is an interesting subject of debate that was motivated by the initial announcement of Go-Explore. The ability to harness determinism and perform such resets forces us to recognize that there are two different types of

problems we wish to solve with RL algorithms: those that require stochasticity at test time only, and those that require stochasticity during both testing and training." This specific idea is very important and may lead to fundamental change in the way we perceive RL today. as it splits RL problems into two categories as mentioned above and we may see different techniques developed to resolve each type of problems. We may ending have two big RL families of algorithms. Go-Explore is so presented as one of the family of problems that require stochasticity only during testing. Of course a lot more has been said and developed in the research paper but we preferred to focus on the points we covered in the discussion part due to the short time between the paper appearance and the deadline.

7 Conclusion

Go-Explore algorithm is a new type of algorithm for hard-exploration problems. The agent explores the environment until it solves the game (First phase) and optionally robustify (Second phase) in order to expand its knowledge of the environment. It opens many new exciting research directions:

- Using Go-Explore in other contexts.
- Hybridizing insights from Go-Explore with other RL algorithms.

We explained how the algorithm works, and how it is made to solve high dimension states space with sparse rewards. The key insight here is the ability to get back to visited states to allow further exploration. After that comes the robustification step as we improve the found policy in a kind of an automated imitation learning procedure. We also talked about some points left unclear in the blog post such as the ability to extend the use of the algorithm over other RL problems especially in stochastic environment and we talked about the possible new perspectives behind Uber researchers' released paper.

At the same time we have used an existing notebook that has the exploration step of the Go-Explore and tested it on another game. The results seem to be conclusive. Finally, we believe that the story is not over yet. Many publications are yet to come in order to verify those results, improve performance using the proposed solutions and the discussion that Go-Explore initiated will have significant implications on all the RL field.

References

[1] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, Jeff Clune : *Go-Explore: a New Approach for Hard-Exploration Problems.* arXiv:1901.10995

[2] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley and Jeff Clune: Montezuma's Revenge Solved by Go-Explore, a New Algorithm for Hard-Exploration Problems (Sets Records on Pitfall, Too)
`https://eng.uber.com/go-explore/`

[3] Tim Salimans, Richard Chen *Learning Montezuma's Revenge from a Single Demonstration.* arXiv:1812.03381.

[4] Yusuf Aytar, Tobias Pfaff, David Budden, Tom Le Paine, Ziyu Wang and Nando de Freitas *Playing hard exploration games by watching YouTube.* arXiv:1805.11592.

[5] Arthur Juliani: On "solving" Montezuma's Revenge,
https://medium.com/@awjuliani/on-solving-montezumas-revenge-2146d83f0bc3

[6] *https://github.com/R-McHenry/ParallelizedGoExplore*