Go-Explore: a New Approach for Hard-Exploration Problems

Kiet Lorenzo Truong

Proseminar Künstliche Intelligenz im WS 2022
kiet.truong@uni-ulm.de

Abstract. The Go-Explore algorithm is a reinforced learning approach designed specifically for difficult exploration problems or environments where it is particularly difficult to gather information and learn about the environment. The algorithm works by first exploring the environment extensively to identify and archive promising states, and then using exploitation to perform actions that are likely to lead to informative or rewarding states. The algorithm also includes a robustification phase in which it tests and improves the robustness of the solution, often using imitation learning. Overall, the Go-Explore algorithm offers a number of advantages for tackling difficult exploration problems, including its ability to effectively explore and solve complex environments and its ability to perform well on a range of tasks. However, it also has some limitations, such as its potential need for large amounts of computational resources and its limited use of domain knowledge.

1 Introduction

1.1 Definition of reinforcement learning

Reinforcement learning (RL) is a subfield of machine learning that deals with the problem of an agent learning to act in an environment so as to maximize a cumulative reward signal. RL is an example of trial-and-error learning, where an agent learns to improve its behavior by interacting with the environment and receiving feedback in form of rewards or penalties. The agent's goal is to learn a policy that maximizes the expected cumulative reward over time. In some cases, the agent might encounter reinforcement learning problems that can be considered challenging when the environment is complex or the rewards are sparse and deceptive.Sparse and deceptive rewards are "hard exploration" problems where the rewards are infrequent or difficult to interpret. In environments with sparse rewards, it can be challenging for an agent to learn an optimal policy because the rewards it receives for its actions are infrequent and may not provide enough information about the consequences of those actions. Deceptive rewards are rewards that are misleading or difficult to interpret, and they can make it even harder for the agent to learn an optimal strategy, and furthermore, they can cause the agent to get stuck in local optima. For example, an agent that is trying to navigate through a maze might receive a reward for reaching a dead end, which could be deceptive because it leads the agent away from the goal.

Markov Decision Process (MDP) is a mathematical framework for modeling decision making problems and is commonly used to formally describe RL problems. MDPs are defined by the set of states (S), set of actions (A), transition function (T(S,A,S')), reward function (R(S,A,S')) and trajectories. In an MDP, the states represent all possible situations that the agent can be in, and the actions represent the set of possible decisions the agent can make. The transition function T(S,A,S') defines the probability of transitioning from state S to state S' after taking action A. The reward function R(S,A,S') assigns a scalar reward to each state-action-state tuple (S,A,S'). It represents the immediate reward the agent receives after transitioning from state S to S' by taking action A, and gives information about the goal of the problem and how well the agent is doing. A trajectory refers to the sequence of states, actions, and rewards that the agent experiences as it follows its policy through time. It is a representation of the agent's experience in the environment. These components together provide a way to model decision-making problems and allow the use of RL algorithms to find optimal solutions.

An example of an RL problem modeled as an MDP is a monkey trying to collect bananas in a jungle. The states in this MDP could be the different locations in the jungle where the monkey could be. The actions could be the different directions the monkey can move. The rewards could be the number of bananas the monkey collects at each location. The monkey starts in a random location in the jungle. At each time step, the monkey chooses an action based on its current state and the environment transitions to a new state. The monkey receives a reward for the number of bananas it collects at the new location. The goal of the monkey is to learn a policy that maximizes the total number of bananas collected over time.

1.2 Overview of the Go-Explore algorithm

Go Explore was developed by researchers at Uber Ai Labs and it was first described in a paper published in 2019. The paper was titled ”Go Explore: A New Approach for Hard-exploration Problems,” and it was authored by Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley and Jeff Clune.

The Go-Explore algorithm is a reinforcement learning (RL) algorithm that is designed to solve hard exploration problems, where the state space is large and the rewards are sparse. These types of problems can be challenging for traditional RL algorithms, as they may become stuck in suboptimal policies or local optima and miss out on better opportunities for learning and improvement.

In addition, the algorithm was developed to improve the efficiency and effectiveness of learning and to overcome some challenges that traditional reinforcement learning algorithms face when dealing with sparse rewards in an environment by using a ”first return, then explore” strategy to guide the exploration process and help the agent learn and make progress in its environment. ”first return, then explore” refers to the strategy of first focusing on returning to previously visited states that have been found to be promising, and then using those states as a starting point for further exploration. By returning to previously visited states that have been found to be promising, the algorithm can discover new, related states that may not have been discovered on the first visit. This can help the agent learn more about the environment and improve its performance.

The main idea behind Go-Explore is to identify and return to ”promising” states, which are states that have led to high reward in the past. The algorithm keeps track of these states in a separate archive and uses them as a starting point for further exploration. Go-Explore has shown promising results in a variety of environments and has been able to solve some Atari games that were previously thought to be unsolvable using traditional exploration techniques. It is an interesting and innovative approach to exploration that is worth considering in the context of solving hard exploration problems.

1.3 Hard-exploration problems

Hard-exploration problems are rewards which are sparse or deceptive. Go-Explore hypothesized that the causes of these problems are rooted in the concepts of ”detachment” and ”derailment.”

”Detachment” refers to the idea that an agent, in an Reinforcement learning problem, driven by intrinsic motivation (IM) can become detached from the frontiers of high intrinsic reward (IR). In IM-based RL, the agent learns by exploring the environment and seeking out states that have high intrinsic rewards. However, when an agent discovers multiple areas of the state space that produce high IR, its policy may focus on one such area. Once it has exhausted the IR offered by that area, it may struggle to rediscover the frontier it detached from, as it has already consumed the IR that led to that frontier. This can lead to a problem where the agent becomes ”detached” from high-reward areas of the state space, and may not be able to effectively explore and learn from these areas in the future. This can be detrimental to the agent's learning process, as it may miss out on important opportunities for learning and improvement.

In the example of the monkey trying to collect bananas in a jungle, detachment refers to the idea that the monkey may become detached from areas of the jungle where it had previously found high numbers of bananas. For example, let's say that the monkey initially discovers a dense area of the jungle with many bananas. The monkey may focus on this area and collect as many bananas as possible, but after some time, the number of bananas in that area may decrease. The monkey's policy may then switch to exploring another area of the jungle, where it may also find bananas. However, once the monkey exhausts the bananas in the second area, it may struggle to return to the first area where it had initially found many bananas. This struggle to return to the first area can be due to the fact that the monkey has already consumed the intrinsic reward (IR) that led to that frontier, and it likely will not remember how to return to that frontier due to catastrophic forgetting. Additionally, it's possible that there may be a dearth of remaining IR near the areas of state space visited by the current policy, and therefore, no learning signal remains to guide the monkey to further explore in an effective and informed way.

”Derailment” is a term used to refer to the difficulty an agent has in returning to the promising states it has discovered. This can happen when the environment is stochastic and not always the same, and the agent must perform a series of actions to reach the promising state. Some of these actions may be riskier and have a lower or negative reward. This may make it more difficult for the agent to return to the promising state, as it may be deterred by the possible negative rewards along the way. This may cause the agent to ”derail” from its exploration path and be unable to return to the promising state it had discovered.

In the scenario where the monkey is trying to collect bananas in a jungle, the monkey's strategy may shift to exploring another area of the jungle where it can also find more bananas if the bananas in the previous area become fewer. However, the monkey may have trouble returning to the first area, as the environment is uncertain and not always the same when the outcome of an action is not known in advance, or the state of the environment changes over time. In this case the environment may be uncertain because the location of the bananas may change from one day to the next, or the number of bananas in a specific area may vary. Additionally, the environment may be uncertain because of the presence of other animals that may steal the bananas or the weather may affect the growth of bananas. This means that the monkey can not rely on previous experience to know where to find bananas or how many bananas will be in a specific area. The monkey may have to execute a series of actions to reach the previous area, some of which may be more risky or may have lower or negative rewards. This can make it challenging for the monkey to return to the previous area, as it may be deterred by the prospect of negative rewards along the way. This can cause the monkey to become ”derailed” from its exploration path and unable to return to the promising state it had discovered.

2 Exploration and Exploitation in reinforcement learning

2.1 Definition of exploration

In reinforcement learning, exploration refers to the process of an agent learning about and navigating its environment in order to achieve its goals. This can involve taking actions that may not be immediately rewarding, but that have the potential to improve the agent's understanding of the environment and lead to better long-term performance. Exploration is a crucial aspect of RL because it allows agents to discover new states, actions, and rewards that may not be immediately apparent or accessible. It is often necessary in environments where the rewards are sparse or the state space is large, as it allows the agent to learn about and navigate these environments effectively.

Exploration can be particularly challenging in environments where the rewards are sparse or the state space is large, as the agent may need to take many steps or visit many states before finding a rewarding one. In such cases, an effective exploration strategy is crucial to the agent's ability to learn and make progress. There are several approaches to exploration in RL, including random exploration, directed exploration, and intrinsically motivated exploration. The choice of exploration strategy will depend on the specific characteristics of the environment and the goals of the agent.

2.2 Definition of Exploitation

In contrasted with exploration, exploitation is defined as a process of selecting actions that are expected to lead to the highest rewards based on current knowledge about the environment. This is done with the goal of maximizing reward and making progress toward the agent's goals.

Exploitation is important in RL because it allows the agent to make progress towards its goals and maximize its reward. It is particularly important when the agent has learned a good policy for the current environment and the rewards are dense, as it allows the agent to efficiently follow the policy and maximize its reward. However, exploitation can also be a double-edged sword, as it can cause the agent to become stuck in a suboptimal policy or in a local optimum. In such cases, exploration may be necessary to help the agent discover better policies and escape suboptimal states.

In the context of the Go-Explore algorithm, exploitation refers to the process of using the states in the archive (which represent promising areas of the state space) as a starting point for further exploration. The Go-Explore algorithm uses exploitation to guide the exploration process towards regions of the state space that are more likely to be rewarding, based on the agent's past experience. This is in contrast to random exploration, which involves randomly selecting actions and observing the resulting rewards and states, and directed exploration, which involves using prior knowledge or heuristics to guide the exploration process.

2.3 Problems with exploration and exploitation

One of the main problems with exploration and exploitation in reinforcement learning is the tradeoff between exploration and exploitation. If the agent exploits too much, it may get stuck in a suboptimal policy and miss better opportunities. On the other hand, if the agent explores too much, it may not make sufficient progress toward its goals.

There are several approaches that can be used to resolve the tradeoff between exploration and exploitation, including Epsilon-Greedy exploration, Thompson sampling, and Upper Confidence Bound (UCB) methods. These approaches allow the agent to balance exploration and exploitation by adjusting the probability of exploration or accounting for uncertainty in reward.
One approach that has recently gained attention is the Go-Explore algorithm, which balances exploration and exploitation by using a combination of both. The way the algorithm works is that the agent returns to the previously visited state and starts exploring from there. During exploration, agents record the states they visit and the actions they take to get there. Go-Explore then uses this information to identify promising areas of the environment to explore further, while exploiting the knowledge it has gathered. In this way, the algorithm can explore efficiently and effectively while leveraging exploitation to maximize rewards.

3 How Go-Explore works

The Go Explore algorithm consists of two phases. The first phase is the exploration phase, which is a loop where the agent focuses heavily on exploring the state. In this step, the agent is initialized and an initial state associated with promising cells is selected from the archive for the agent to explore from. Next, theagent then returns to this state and performs random actions in the environment to explore the state space and collect information about it. During exploration, the agent records the states visited and the actions performed along the way. Then Go-Explore updates the archive with new states and better solution to get to previous states. The archive consists of the states and solutions of the cell with the solution visit times and the rewards achieved. In addition, Go-Explore uses the recorded information to identify promising areas of the environment to explore further. These areas are referred as ”cells”. Finally, Go-Explore uses exploitation to take advantage of the knowledge accumulated so far and to maximize rewards.

The algorithm repeats the first phase until the environment is solved or a satisfactory level of performance is achieved. This allows the algorithm to gain initial understanding of the environment and identify promising areas to focus on, using the properties of determinism and resettability to facilitate this process. This contributes to the algorithm's ability to effectively explore and solve the environment and find solutions that may be difficult for traditional reinforcement learning algorithms to discover.

The second phase is ”robustify”. The ”robustify” phase refers to the process of testing and improving the robustness of a solution to the environment. This phase is necessary when the algorithm has found a solution that works well under certain conditions, but may be prone to failure under other conditions. In the Robustify phase, the algorithm tests the solution under various conditions to ensure that it is robust and reliable. This may mean testing the solution in different parts of the environment, with different starting conditions, or under different environmental conditions.
The robustify phase is necessary because it allows the Go-Explore algorithm to ensure that the solution it has found is reliable and robust, and to improve the solution if necessary. By testing the solution in a variety of different conditions, the algorithm can identify any weaknesses or vulnerabilities in the solution and make improvements as needed. Thus, the robustify phase is an important part of the Go-Explore algorithm, as it allows the algorithm to ensure that the solution it has found is reliable and robust, and to improve the solution if necessary. This helps to ensure that the solution is able to effectively solve the environment under a variety of different conditions.

3.1 Cell representations

Cells refer to areas of the environment that are considered particularly informative or rewarding. The algorithm divides the state into cells to organize and prioritize the exploration process. By dividing the state into cells, Go-Explore is able to focus its exploration efforts on the most promising areas of the environment and increase its chances of finding a solution. This makes exploration and solution more manageable.

One of the advantages of dividing the state space into cells is that Go-Explore can reduce the dimensionality of the state space, which can make it easier to explore and solve. High-dimensional state spaces can be difficult to explore and solve for several reasons, including the larger number of possible states and the higher computational cost. With a large number of possible states, it is more difficult to explore the entire state space and accumulate enough information to effectively solve the environment. This can lead to slower convergence and lower performance. In addition, high-dimensional state spaces may also require more computational resources to explore and solve, which can limit the scalability of the algorithm.

On the whole, dividing the state space into cells is an important part of the Go-Explore algorithm, as it allows the algorithm to focus its exploration efforts on the most promising areas of the environment and reduce the dimensionality of the state space. This allows Go-Explore to explore efficiently and effectively, and to find solutions that may be difficult for traditional reinforcement learning algorithms to discover.

3.2 Domain knowledge

Go-Explore is a reinforcement learning algorithm that does not explicitly require domain knowledge to function. It is designed to be able to solve a wide range of environments with little or no prior knowledge about the structure of the environment or the characteristics of the actions available to the agent. This is one of the key advantages of Go-Explore, as it allows the algorithm to be applied to a wide range of problems without requiring domain-specific knowledge. This makes it a useful tool for solving problems in areas where domain knowledge may be limited or unavailable.

However, there are some potential disadvantages to this approach. First, domain knowledge can often be a valuable tool for guiding exploration and improving the efficiency of the learning process. By explicitly incorporating domain knowledge into the algorithm, it may be possible to more effectively explore the environment and gather the information needed to solve it. Second, in some cases, domain knowledge may be necessary to understand the structure of the environment and identify promising areas to focus on. Without domain knowledge, it may be more difficult for the algorithm to effectively explore and solve the environment.

One potential disadvantage of using domain knowledge is that it may not always be available or may be difficult to obtain. This can limit the applicability of the algorithm in certain situations. Additionally, domain knowledge may also be specific to a particular problem or environment, which can make it difficult to generalize to other situations.

3.3 How the agent returns to the cell

There are several ways in which an agent in Go-Explore can return to a state from the archive, ideally when the environment consists of ”determinism” and ”resettable”. ”Determinism” refers to the property of an environment in which the result of an action is predictable and the same action always leads to the same result. "Resettable" refers to the property of an environment in which the agent can return to a previous state by resetting the environment. When an agent returns to a state in a deterministic resettable environment, it means that the agent is able to revisit a state that it previously visited by resetting the environment and performing the same action or sequence of actions that it previously performed. This can be useful to acquire more information about the state or to leverage knowledge gathered during previous visits.

In a deterministic environment, the agent can be confident that the same action will always lead to the same outcome. This allows the agent to learn from its experiences and make more informed decisions about which actions to take in the future. For example, if the agent takes an action and is rewarded, it can be confident that taking the same action again in the future will also lead to a reward. On the other hand, in a non-deterministic environment, the outcome of an action may vary from one instance to the next, making it more difficult for the agent to learn and adapt. This can make it more challenging for the Go-Explore algorithm to solve the environment and find a satisfactory solution.

In a resettable environment, the agent can revisit previous states by resetting the environment and taking the same action or sequence of actions that it took previously. This allows the agent to gather more information about the state and improve its understanding of it. For example, if the agent takes an action and is rewarded, it can reset the environment and take the same action again to confirm that the reward is consistent. Compared to a non-resettable environment, the agent is unable to revisit previous states, which can make it more difficult for the algorithm to gather information and improve its understanding of the environment.

As a result, determinism and resettability are important properties of an environment that make it easier for the go-explore algorithm to gather more information and improve its understanding of the environment and find a satisfactory solution to the environment.

3.4 When is the environment solved

In the Go Explore algorithm, an environment is considered solved when a satisfactory level of performance has been achieved. A "satisfactory level of performance" refers to the point at which the algorithm has reached a level of performance that is considered acceptable. This point can be determined based on a variety of factors, including the specific goals of the algorithm, the constraints of the environment, and the available resources.

For example, if the goal of the Go-Explore algorithm is to find a solution to an environment that maximizes reward, a satisfactory level of performance may be defined as the point at which the algorithm has found a solution that achieves a certain level of reward. Alternatively, if the goal of the algorithm is to find a solution that is efficient in terms of the number of steps required, a satisfactory level of performance may be defined as the point at which the algorithm has found a solution that requires a certain number of steps or less. The point at which a satisfactory level of performance is achieved will depend on the specific goals and constraints of the algorithm and the environment. Once this point is reached, the Go-Explore algorithm will stop running and the solution will be considered satisfactory.

3.5 Robustify

Go-Explore robustifies the best solutions from the archive through imitation learning. One way Go-Explore can make a solution robust through imitation learning is by using a trained imitation learning model to guide the exploration process.

Imitation learning is a type of machine learning that involves training a model to imitate the behavior of a human or an expert in a particular task. In the context of Go-Explore, an imitation learning model could be trained to mimic the actions of a human or expert in solving the environment. During the robustify phase, the Go-Explore algorithm can use the imitation learning model to guide its exploration process and test the robustness of the solution. For example, the algorithm could use the imitation learning model to take actions in the environment and gather information about the states it encounters. This information can then be used to identify any weaknesses or vulnerabilities in the solution and make improvements as needed.

Using imitation learning to robustify a solution in Go-Explore can be an effective way to improve the robustness and reliability of the solution. By using a trained imitation learning model to guide the exploration process, the algorithm can gather more information about the environment and identify any weaknesses or vulnerabilities in the solution, allowing it to make improvements as needed.

4 Evaluation of Go-Explore

4.1 Comparison to other reinforcement learning algorithms

Compared to other reinforcement learning algorithms, the Go Explore algorithm is particularly well-suited for tackling hard exploration problems. Many traditional reinforcement learning algorithms rely on trial and error and exploration of the environment to gather information and learn about the environment, as this is a key component of the reinforcement learning process. In reinforcement learning, an agent learns by interacting with the environment and receiving feedback in the form of rewards or punishments. Through trial and error, the agent learns which actions are likely to lead to positive outcomes (rewards) and which actions are likely to lead to negative outcomes (punishments). By exploring the environment and gathering information about the conditions it encounters and the actions it takes, the agent is able to learn about the environment and improve its decision making over time. However, in difficult exploration problems, it can be difficult for these algorithms to gather enough information to learn effectively because the environment may be large, complex, or dynamic.

The Go-Explore algorithm addresses this challenge by using a combination of exploration and exploitation to effectively explore the environment and collect information. It also uses strategies such as determinism and resettability to facilitate exploration and improve understanding of the environment. In addition, the Go-Explore algorithm includes a Robustify phase, which is used to test and improve the robustness of the solution to the environment. This enables the Go-Explore algorithm to effectively explore and solve difficult exploration problems that are difficult for traditional reinforced learning algorithms.

4.2 Limitations of Go-Explore

There are a few limitations of the Go-Explore algorithm that are worth considering:

- There are the computational resources. The Go Explore algorithm can require a large amount of computational resources because it involves extensive exploration and testing of the solution. This can be particularly challenging in environments where the cost of interacting with the environment is high, or where the environment is particularly large or complex.
- Go-Explore relies on the properties of determinism and resettability to facilitate exploration and improve understanding of the environment. However, these properties are not always present in real-world environments, which could limit the effectiveness of the algorithm in certain situations.
- Another limitation is the limited use of domain knowledge. While the Go-Explore algorithm can use domain knowledge when it is available, it does not explicitly require domain knowledge to function. As a result, it may not be as effective as other algorithms specifically designed to use domain knowledge.

All in all, while the Go-Explore algorithm has many strengths and offers a number of advantages for tackling difficult exploration problems, it also has some limitations that should be considered when deciding whether it is the best choice for a particular reinforcement learning task.

5 Applications of Go-Explore

5.1 Examples of successful use cases

The Go-Explore algorithm has been applied to a number of different environments and has demonstrated success in a variety of hard exploration problems. Some examples of successful use cases for the Go-Explore algorithm include:

1. The Go-Explore algorithm has been applied to the Atari 2600 game Montezuma's Revenge and has demonstrated success in solving this complex and challenging environment. In fact, the algorithm set a record by beating both the human world record and past reinforcement learning records on the game. Montezuma's Revenge is a particularly challenging environment for reinforcement learning algorithms because it is large, complex, and has a high level of uncertainty. The Go-Explore algorithm was able to effectively explore and solve the environment, demonstrating its effectiveness in hard exploration. problems.
2. The Go Explore algorithm was also applied to a number of robotic tasks, such as sorting items into the shelf, and was able to perform well in these tasks. In these environments, the algorithm has been able to effectively explore and learn a control policy that enables it to successfully complete the task.

5.2 Transfer learning as a potential future development

Transfer learning is a machine learning approach that involves using the knowledge and experience gained from solving one task to improve the performance on a related task. Transfer learning can be particularly useful when there is a shortage of data or resources available to learn a new task from scratch.

One way in which the Go-Explore algorithm could be improved through the use of transfer learning is by using the knowledge and experience gained from solving one task to guide the exploration process on a related task. For example, the algorithm could use the knowledge it has gained about the structure of the environment and the types of states that are likely to be most informative or rewarding to explore on one task to guide its exploration on a related task, like sorting objects into a shelf in robotics.

This could allow the algorithm to more effectively gather the information it needs to solve the new task and improve its performance. In addition, the Go-Explore algorithm could use transfer learning to adapt more quickly to new tasks or environments by leveraging its previous learning experiences. By using the knowledge and experience gained from solving one task to guide its exploration on a new task, the algorithm could more effectively explore and learn in the new environment, improving its efficiency and performance.

Overall, the use of transfer learning could potentially improve the performance of the Go-Explore algorithm by allowing it to more effectively use its previous learning experiences to guide its exploration and adapt to new tasks or environments.

6 Conclusion

Go Explore is a reinforcement learning algorithm that has been shown to be effective at achieving high levels of performance on a variety of Atari 2600 games. One key feature of Go Explore is its ability to handle determinism, which refers to the property of a system that exhibits the same behavior each time it is run under the same conditions. Go Explore is able to effectively handle determinism in video game environments by using agents to explore the game state space and by focusing on return-based exploration, which prioritizes visiting states that have previously resulted high rewards. Another key feature of Go Explore is its ability to be reset, which refers to the ability of the algorithm to return to a previous state or starting point. This is important because it allows Go Explore to explore the game state space more thoroughly, as it can return to previously visited states and try different paths or strategies. In addition, Go Explore has been shown to be robust, which means that it is able to perform well even in the face of uncertainty or variability. This is important because it allows Go Explore to be used in a wide range of environments and to adapt to changes in the environment over time.

Finally, there are many possibilities for further research on Go Explore in terms of future directions. One possibility is to investigate how the algorithm can be applied to other types of hard exploration problems, such as those that occur in real-world environments or in more complex and realistic video games. It would also be interesting to see how Go Explore can be combined with other machine learning techniques, such as unsupervised learning or transfer learning, to enhance its performance. Finally, it will be important to continue to analyze the properties of Go Explore and to understand its limitations in order to identify ways in which it can be made more effective. Overall, Go Explore is a promising tool for tackling hard exploration problems, and further research on this algorithm has the potential to yield valuable insights and advances in this field.