ATAI

REINFORCEMENT LEARNING IN MINECRAFT

September 29, 2020

Authors:

Erik Angulo

Xabier Lahuerta

Oihane Roca

Contents

1 Contextualization

The purpouse of this paper is to explain how agents can perform actions in Minecraft using reinforcement learning algorithms.

1.1 What is Minecraft?

Minecraft is a sandbox video game [1] developed by Mojang Studios. In Minecraft, players explore a blocky, procedurally-generated 3D world with infinite terrain, and may discover and extract raw materials, craft tools and items, and build structures or earthworks. Depending on game mode, players can fight computer-controlled "mobs" [2], as well as cooperate with or compete against other players in the same world. Game modes include a survival mode, in which players must acquire resources to build the world and maintain health, and a creative mode, where players have unlimited resources. Players can modify the game to create new gameplay mechanics, items, and assets.

Figure 1: Minecraft Logo.

[1] Sandbox games are often associated with open world concepts which gives the player freedom of movement and progression in the game's world.

[2] A mob, short for mobile, is a computer-controlled non-player character (NPC) in a computer game.

1.2 Reinforcement Learning (RL)

In a nutshell, RL is the study of agents and how they learn by trial and error. It formalizes the idea that rewarding or punishing an agent for its behavior makes it more likely to repeat or forego that behavior in the future.

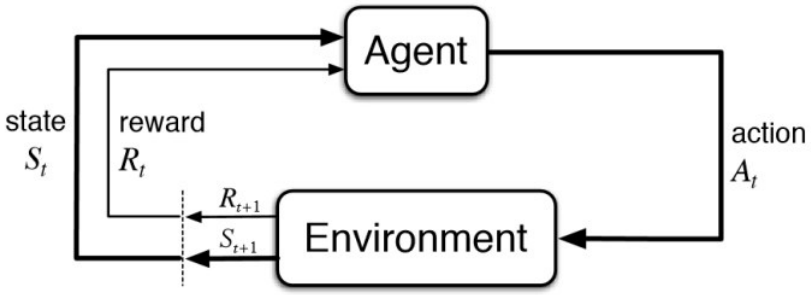

Figure 2: The standard reinforcement learning model.

1.3 MarlÖ

MarlÖ is an API built on top of Project MalmÖ made by Microsoft. MalMÖ, for his part, is a platform to experiment and research with Artifical Intelligence which is built on top of Minecraft. As a result, it can communicate directly with Java Minecraft and that's how the program manages to control the Minecraft player. Malmo's APIs gives the possibility to access actions, observations (i.e. location, surroundings, video frames, game statistics) and other data that Minecraft provides. These data is then used to build more friendly and higher level APIs, such as the ones used at the previously mentioned MarlÖ and MineRL.

1.4 MineRL

MineRL, the brainchild of a team of researchers from Carnegie Mellon University, tackles an ambitious problem facing the machine learning community: an increasing demand for large amounts of computational resources to replicate state-of-the-art research.

In short MineRL consists of several major components:

- **MineRL-v0 Dataset -** One of the largest imitation learning datasets with over 60 million frames of recorded human player data. The dataset includes a set of environments which highlight many of the hardest problems in modern-day Reinforcement Learning: sparse rewards and hierarchical policies.
- **minerl -** A rich python3 package for doing artificial intelligence research in Minecraft. This includes two major submodules.
  - *minerl.env* - A growing set of OpenAI Gym environments in Minecraft. These environments leverage a synchronous, stable, and fast fork of Microsoft Malmo called MineRLEnv.
  - *minerl.data* - The main python module for ext with the MineRL-v0 datase.

1.4.1 MineRL 2020

MineRL 2020 is the fourth competition based on Project Malmo, an experimentation platform using Minecraft to advance AI.

In the competition, participants develop a system to obtain a diamond in Minecraft using only four days of training time. A key challenge in making competitions more accessible to people with different levels of interest, expertise, and resource access is the preparation of a good set of baselines they can use to ramp up the task and environment and leverage in their solutions.

The company's intensive work resulted in an extensive set of excellent baselines, which utilized its deep learning framework Chainer and included behavioral cloning, Deep Q-learning from Demonstrations (DQfD), Rainbow, and proximal policy optimization (PPO).

2 Reinforcement Learning

The main characters of Reinforcement Learning are the agent and the environment. The environment is the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.

The agent also perceives a reward signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called **return**. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.

2.1 States and observations

A state (denoted as s) is a complete description of the state of the world. There is no information about the world which is hidden from the state. An observation (denoted as o) is a partial description of a state, which may omit information.

States and observations are almost always represented by a real-valued vector, matrix, or higher-order tensor. For instance, a visual observation could be represented by the RGB matrix of its pixel values. The MineRLTreechop-v0 environment's observation space is as follows:

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

pov holds a $64x64x3$ sized matrix, containing the RGB value of each pixel on the $64x64$ window the agent sees.

2.2 Action spaces

Different environments allow different kinds of actions. The set of all valid actions in a given environment is often called the action space.

Some environments have discrete action spaces, where only a finite number of moves are

available to the agent, whereas other environments have continuous action spaces in which actions are real-valued tensors.

2.3 Policies

A policy is a rule used by an agent to decide what actions to take. It can be deterministic.

$$a_t = \mu(s_t)$$

Or it may be stochastic.

$$a_t \sim \pi(\cdot|s_t)$$

2.4 Trajectories

A trajectory τ is a sequence of states and actions in the world.

$$\tau = (s_0, a_0, s_1, a_1, \ldots)$$

The very first state of the world, s_0, is randomly sampled from the start-state distribution, sometimes denoted by ρ_0:

$$s_0 \sim \rho_0(\cdot)$$

State transitions (what happens to the world between the state at time t, s_t, and the state at $t+1$, s_{t+1}), are governed by the natural laws of the environment, and depend on only the most recent action, a_t. They can be deterministic

$$s_{t+1} = f(s_t, a_t)$$

or stochastic.

$$s_{t+1} \sim P(\cdot|s_t, a_t)$$

Actions come from an agent according to its policy. Trajectories are also frequently called episodes.

2.5 Reward and return

The reward function R is critically important in reinforcement learning. It depends on the current state of the world, the action just taken, and the next state of the world:

$$r_t = R(s_t, a_t, s_{t+1})$$

One kind of return is the *finite-horizon undiscounted return*, which is just the sum of rewards obtained in a fixed window of steps:

$$R(\tau) = \sum_{t=0}^{T} r_t$$

Another kind of return is the *infinite-horizon discounted return*, which is the sum of all rewards ever obtained by the agent, but discounted by how far off in the future they're obtained. This formulation of reward includes a discount factor $\gamma \in (0, 1)$:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

An infinite-horizon sum of rewards may not converge to a finite value, but with a discount factor and under reasonable conditions, the infinite sum converges.

2.6 Reinforcement Learning problem

The goal in RL is to select a policy which maximizes expected return when the agent acts according to it. Let's suppose that both the environment transitions and the policy are stochastic. In this case, the probability of a T -step trajectory is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$

The expected return (for whichever measure), denoted by $J(\pi)$, is then:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{\mathbf{E}}[R(\tau)]$$

The central optimization problem in RL can then be expressed as finding the optimal policy π^* as:

$$\pi^* = arg\max_{\pi} J(\pi)$$

3 Environments

The agents can achieve multiple objectives. For each objective, a different environment will be loaded into Minecraft. The environment will have specific objects and terrains in order to make the agent objectives achievable.

Besides, an XML file can be created specifying environment attributes, for example, where the agent will be loaded. Rewards can be set as well, positive points mainly to the mission we want the agent to complete whereas negative points are given to actions that won't help succeding, such as dying. The documentation related the XML Schema can be found here.

3.1 MarlÖ

In the case of MarlÖ, the agents can be trained to achieve missions based on the following environments:

- **MarLo-MazeRunner-v0**: the environment is formed by flat purple material with white cubes. The agent should follow the cubes to arrive to the final point of the maze.
- **MarLo-CliffWalking-v0**: there is some lava in the lava as well as some safe terrain. The agent should arrive to destination withour falling in the lava.
- **MarLo-CatchTheMob-v0**: the environment has some obstacles and a mob. The agent will try to catch the mob.
- **MarLo-FindTheGoal-v0**: it is a closed room with a yellow cube. The agent will try to find it.
- **MarLo-Attic-v0**: the agent should arrrive to the goal which is at the top of the loaded map. In order to do that, it must learn go up jumping.
- **MarLo-DefaultFlatWorld-v0**: there is a building in a flat world, and the agent objective will be entering inside.
- **MarLo-DefaultWorld-v0**: the Minecraft default game wolrd will be loaded. In this mode agents can be trained to perform whatever actions desired by the programmer.
- **MarLo-Eating-v0**: this environment has a flat floor with some food items on it. The agent should collect the items as fast as possible.

- **MarLo-Obstacles-v0**: It is the same environment as *MarLo-FindTheGoal-v0*, but in this case the building is bigger, with the possibility of having more rooms and obstacles.
- **MarLo-TrickyArena-v0**: the floor is made by a variety of materials. The agent should find the specified material.
- **MarLo-Vertical-v0**: it is the same environment as *MarLo-FindTheGoal-v0*, but the goal is in a room above the loaded room. The difference with *MarLo-Attic-v0* is that in this case, in order to go up there are some stairs.

3.2 MineRL

The MineRL framework also offer some environments at which agents can be trained to perform some actions and complete objectives, such as the following:

- **MineRLTreechop-v0**: the agent must collect 64 *minercaft:log*. The agent begins in a forest biome (near many trees) with an iron axe for cutting trees.
- **MineRLNavigate-v0**: in this task, the agent must move to a goal location denoted by a diamond block. The agent must find the final goal by searching based on local visual features. This variant of the environment is sparse. In this environment, the agent spawns on a random survival map.
- **MineRLNavigateExtreme-v0**: this task's goal is the same as in the *MineRLNavigate-v0*, but in this environment the agent spawns in an extreme hills biome.
- **MineRLNavigateDense-v0**: this task's goal is the same as in the *MineRLNavigate-v0*, but this variant of the environment is dense reward-shaped where the agent is given a reward every tick for how much closer (or negative reward for farther) the agent gets to the target.
- **MineRLNavigateExtremeDense-v0**:this task is the same as the *MineRLNavigateDense-v0*, but in this environment the agent spawns in an extreme hills biome.
- **MineRLObtainDiamond-v0**: in this environment the agent is required to obtain a diamond. The agent begins in a random starting location, on a random survival map, without any items. This variant of the environment is sparse.
- **MineRLObtainDiamondDense-v0**: this task's goal is the same as in *ObtainDiamond*, but this variant of the environment is dense reward-shaped where the agent is given a

reward every tick for how much closer (or negative reward for farther) the agent gets to the target.

- **MineRLObtainIronPickaxe-v0**: in this environment the agent is required to obtain an iron pickaxe. The agent begins in a random starting location, on a random survival map, without any items. This variant of the environment is sparse.
- **MineRLObtainIronPickaxeDense-v0**: this task's goal is the same as in *ObtainIronPickaxe*, but this variant of the environment is dense reward-shaped where the agent is given a reward every tick for how much closer (or negative reward for farther) the agent gets to the target.

4 Installation and usage

4.1 MarLÖ

The process to install MarlÖ is the following:

```
conda create python=3.6 --name marlo
conda config --add channels conda-forge
conda activate marlo
conda install -c crowdai malmo
pip install -U marlo
```

All the dependencies needed will also be installed. However, the user does not have to have the Minecraft game (Java version) purchased and installed in order to proceed, because Microsoft made a development adaptation of the game for the purpose of Project Malmo.

4.1.1 Launch Minecraft Client

```
$MALMO_MINECRAFT_ROOT/launchClient.sh -port 10000
```

It will launch the Minecraft game

4.1.2 Create the environment

We set the environment we want. More than one agent can be created as well, using client_pool variable.

```
import marlo
client_pool = [('127.0.0.1', 10000)]
join_tokens = marlo.make('MarLo-MazeRunner-v0',
                         params={
                           "client_pool": client_pool,
                           "agent_names" : ["MarLo-Agent-0"]
                         })
assert len(join_tokens) == 1
env = marlo.init(join_tokens[0])
env.reset()
```

4.1.3 Start the game loop

```
done = False

while not done:
```

```
    _action = env.action_space.sample()
    obs, reward, done, info = env.step(_action)

env.close()
```

In this code above, no reinforcement learning is happening. The code above only samples randomly an action from all the actions available in the environment and performs it. For each action, we get the observation of the agent as well as the reward quantity gotten for performing the action and if the agent finished the objective.

In order to implement a Reinforcement Learning, instead of randomly sampling the action space, the idea would be to perform some kind of policy optimization method.

4.2 MineRL

MineRL is included in the pip python repository, so it is as easy to install as one command:

```
pip install minerl gym
```

4.2.1 Download and building datasets

MineRL is composed of two main components, which are the data and the environments. The data consists of around 65 GB of collected from client side re-simulated Minecraft demonstration data.

In order to download that data, the following command should be run:

```
$MINERL_DATA_ROOT="your/local/path" python3 -m minerl.data.download
```

Once the data has been downloaded, MineRL offers a way of easily building a dataset, as described by the following code piece:

```
data = minerl.data.make(MineRLObtainDiamond-v0')

for current_state, action, reward, next_state, done \
    in data.batch_iter(batch_size=1, num_epochs=1, seq_len=32):

        # Print the POV @ the first step of the sequence
        print(current_state['pov'][0])
```

```
        # Print the final reward of the sequence!
        print(reward[-1])
        # Check if final (next_state) is terminal.
        print(done[-1])
        #Do something with the data!!
```

4.2.2 Creating an agent

Creating an agent in MineRL is the same as creating an agent on a gym-like environment, so that means that the code for creating and taking actions on an agent is very similar as the explained in MarLÖ section. The following piece of code creates an agent in MineRL which just goes forward jumping and attacking, as an example of the simplicity of usage:

```
import minerl
import gym
env = gym.make('MineRLNavigateDense-v0')


obs  = env.reset()
done = False
net_reward = 0

while not done:
    action = env.action_space.noop()

    action['camera'] = [0, 0.03*obs["compassAngle"]]
    action['back'] = 0
    action['forward'] = 1
    action['jump'] = 1
    action['attack'] = 1

    obs, reward, done, info = env.step(
        action)

    net_reward += reward
    print("Total reward: ", net_reward)
```

5 Conclusions

We have drown two clear conclusions from all this research:

- It is a good approach to understand and dive into Reinforcement Learning, as one can understand it and prove the algorithms themselves in a game that is played by millions of people around the world. These kind of frameworks narrow the gap Machine Learning experts and unexpert users, allowing them to acquire knowledge from this Artificial Inteligence area in a more atractive way.

- There is not much documentation on MarlÖ or even the environments related to it. This is mostly because MarLÖ was designed to help people to participate in a certain competition. The people attending the competition were supposed to already have some knowledge about Reinforcement Learning methods. As the tool was designed for that use case, there is not much code available, because the code written by the participants is private so that no other team can use the same techniques. The most useful documentation that can be found for that tool is the frameworks source code itself.

6 References

- Malmotutorial
- MarLÖ Documentation
- MineRL 2020
- MineRL Documentation
- XML Schema
- L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. 1996.
- William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft-demonstrations, 2019.
- Johnson M., Hofmann K., Hutton T., Bignell D. (2016) The Malmo Platform for Artificial Intelligence Experimentation. Proc. 25th International Joint Conference on Artificial Intelligence, Ed. Kambhampati S., p. 4246. AAAI Press, Palo Alto, California USA.
- Diego Perez-Liebana, Katja Hofmann, Sharada Prasanna Mohanty, Noburu Kuno, Andre Kramer, Sam Devlin, Raluca D. Gaina "The Multi-Agent Reinforcement Learning in MalmÖ (MARLÖ) Competition", 2019.
- Alexey Skrynnik, Aleksey Staroverov, Ermek Aitygulov, Kirill Aksenov, Vasilii Davydov,and Aleksandr I. Panov. Hierarchical deep q-network from imperfect demonstrations inminecraft, 2019.