Reinforcement Learning with TF-Agents on MountainCar-v0

Zihan Chen (zc2485), Daijie He (dh2981), Vikki Sui (ks3747), Jinxu Xiang (jx2399)

December 20, 2020

Abstract

Reinforcement learning (RL) is an area of machine learning concerned with action decision to maximize the notion of cumulative reward. Based on this, TF-Agents were first introduced in 2018 to implement decision making. This project will train a specific TF-Agent to fit the MountainCar-V0 Environment in OpenAI Gym environment, which use simple physics rules to solve classic control problems. For this particular agent, it will automaticallt find the trick to push the car left or right, till it is equipped with enough power to climb the hill. Detailed training process and description videos will be provided to help explain this agent.

1 Introduction

Reinforcement learning is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. [1]

Reinforcement learning is not a specific algorithm, but a general term for a class of algorithms. The idea of reinforcement learning algorithm is very simple. Take the game as an example. If a certain strategy can be adopted in the game to get a higher score, then this strategy should be further "strengthened" in order to continue to achieve better results. This strategy is very similar to various "performance rewards" in daily life. We usually use this strategy to improve our game level.

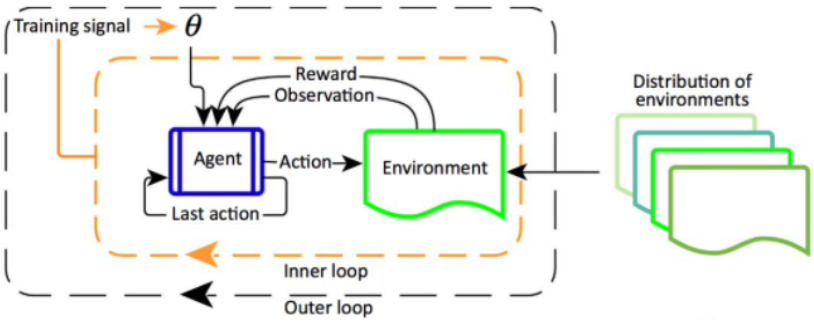

Figure 1: Reinforcement Learning

2 Environment

Our project is to solve MountainCar-v0 problem. The environment is set in Gym from OpenAI website. In this specific control theory problem, a car is on a one-dimensional track, positioned between two "mountains". The goal is to make an under powered car to the top of a hill on the right side. The only admitted interference toward the car is to push it left or right with a fixed momentum. Thus, we use tf.agents to implement reinforcement learning on the model to let a car drive back and forth automatically to reach the top of the hill.

After thoroughly researching MountainCar-v0 Environment, we modified part of the environment. First, we modified the reward in the original environment to make training more convenient. Second, we modified the route of the car to complicate the problem. We will show the results separately at the end. The figure below shows the original environment [2] and the modified environment.

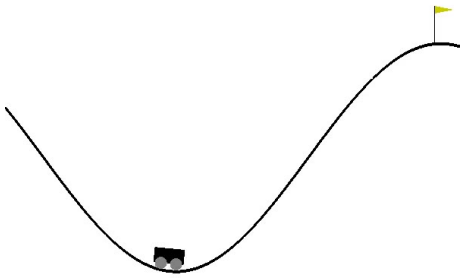

Figure 2: Original Environment

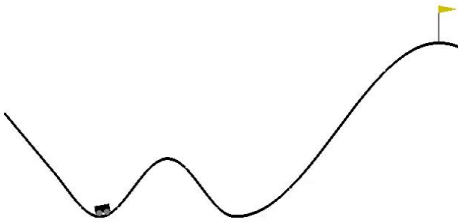

Figure 3: Modified Environment

2.1 Observations

Original Environment [2]:
Starting State: The position of the car on the x-axis is assigned a uniform random value in $[-0.6, -0.4]$.
Starting Velocity: The velocity of the car is always assigned to 0 at the beginning.
Episode Termination: The car position is more than 0.5, which means it has already reached the hilltop. Or episode length is greater than 200.
Route Formula [3]: The position in this environment corresponds to x, and the height corresponds to y. Height does not belong to the value in observations, it corresponds to the position one by one through the following formula:

$$y = sin(3x) \tag{1}$$

Modified Environment:
Starting State: The position of the car on the x-axis is assigned a uniform random value in $[-0.8, -0.2]$.
Starting Velocity: The velocity of the car is always assigned to 0 at the beginning.
Episode Termination: The car position is more than 4.7, which means it has already reached the right hilltop. Or episode length is greater than 500.
Route Formula: The position in this environment corresponds to x, and the height corresponds to y. Height does not belong to the value in observations, it corresponds to the position one by one

through the following formula:

$$y = \begin{cases} 3 * (x + 1.2) * np.cos(-3.6) + np.sin(-3.6) & -2 \leq x \leq -1.2 \\ sin(3x) & -1.2 < x \leq \frac{\pi}{2} \\ -3 * sin(x) + 2 & \frac{\pi}{2} < x \leq 5 \end{cases} \tag{2}$$

Observation	Min	Max
position	-1.2	0.6
velocity	-0.07	0.07

Table 1: Original Observations

Observation	Min	Max
position	-2.0	5.0
velocity	-0.2	0.2

Table 2: Modified Observations

2.2 Actions

The force of both environment are set to $F = 0.001$ and the gravity is set to $G = 0.0025$. If the policy chose to accelerate to left or right, it will change the velocity V by this formula [3]:

$$V = V + (A - 1) * F - \frac{G}{3} * \frac{dy}{dx} \tag{3}$$

where A is the action shown below [2], y and x are the positions.

Num	Action
0	Accelerate to the Left
1	Don't accelerate
2	Accelerate to the Right

Table 3: Actions for both Environment

2.3 Rewards

Original Reward [2]:
The environment will calculate rewards in each step and it will shut down after 200 steps. The final reward is the summation of 200 steps. The original environment will return -1 as reward in each step if the car doesn't achieve the goal. If the car reaches the goal position, the environment will immediately shut down. Thus, the final reward is the negative value of the number of steps required for the car to reach the goal position.

Modified Reward:
Since the reward is not continuous for every state, we need to modify it so that the model can learn different reward values. The changed reward is defined as the reward in the current state minus the reward in the previous state. The reward of each state is the car's momentum

$$R_{mom} = m * G * y + \frac{1}{2} m * V^2 \tag{4}$$

where we set $m = 100$ is the weight of the car. Thus the reward in each step is the increased value of the momentum. If the car achieve the goal, there will be an extra larger reward $R_{goal} = 1$ added to the goal step ensuring that the model can learn the importance of reaching the goal position. We apply this reward to both the original environment and the modified environment. And record the number of steps the car takes to reach the goal position to calculate the original reward.

3 TF-Agents and DQN

TF-Agents makes implementing, deploying, and testing new Bandits and RL algorithms easier. It provides well tested and modular components that can be modified and extended. It enables fast code iteration, with good test integration and benchmarking. [4]

Generally, the process of each machine learning in reinforcement learning is similar to supervised learning, except that the data involved in training is fresh every time. This will result in low data utilization and always learn adjacent parts of the data. Because these data are easy to be strongly correlated, the global optimum cannot be learned. DQN uses the experience playback mechanism, maintains a queue, retains historical Experience (a complete piece of data for learning), and then uses the ϵ-greedy mechanism, a combination of greedy and slightly random sampling. It can extract mini-batch pieces of Experience from the queue for predictive network gradient descent learning. The random sampling part is to ensure the agent's exploration ability.

We treated TF-Agents as a black box, established a DQN agent, saved the data generated according to the agent's policy in the replay buffer, and used it to build a data pipeline to train the model. The detailed steps are shown below:
1. Load the environment into a TimeLimit Wrapper which terminates the game if maximum steps are reached. Then, the results will be wrapped in the TF-Agents handlers. This allows us to set the environment the same as the MountainCar-v0.
2. Create the network, use the Adam optimizer and the DQN Agent. As the heart of a DQN Agent, Q-Network can learn to predict QValues for all actions, given an observation from the environment.
3. Create the replay buffer which can collect data from the environment and use that data to train the agent's neural networks.
4. Create a data pipeline from the replay buffer for training the agent policy.
5. Look at the cycle of training where the experience is drawn from the dataset and used to train the agent.

4 Experiment

4.1 Deep Q-Network (DQN)

In MountainCar-v0, the input data is the observations and the output data is the combination of action and reward. The basic model in DQN we used in TF-Agents is like this:

```
def create_model():
    model = Sequential()
    # input_shape = 2, output_shape = 3
    input_shape = self.env.observation_space.shape[0]
    output_shape = self.env.action_space.shape[0]
    model.add(Dense(48, input_dim=input_shape, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(output_shape, activation='relu'))
    model.compile(loss='mean_squared_error', optimizer=Adam)
    return model
```

4.2 Original Environment with DQN

We trained the agent for 100000 rounds and record the average of rewards and steps of 5 experiments after every 1000 rounds of training.

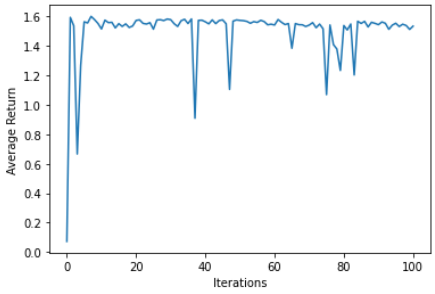

Figure 4: Original Environment Rewards

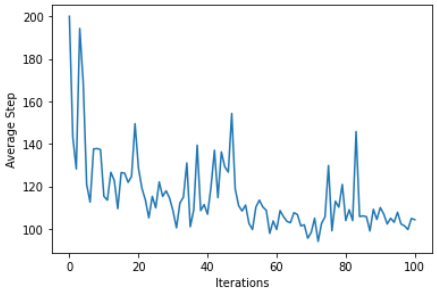

Figure 5: Original Environment Steps

MountainCar-v0 defines "solving" as getting average reward of -110.0 over 100 consecutive trials. [2] Which means "solving" this environment needs the car achieve the goal in 110 steps in average. Our DQN policy can achieve the goal with an average 103.565 steps in 200 consecutive trials. In contrast to this, random policy never reached the goal position in more experiments.

4.3 Modified Environment with DQN

For modified environment, we also trained a new agent for 100000 rounds and record the average of rewards and steps of 5 experiments after every 1000 rounds of training.

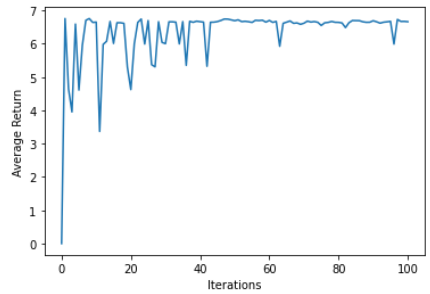

Figure 6: Modified Environment Rewards

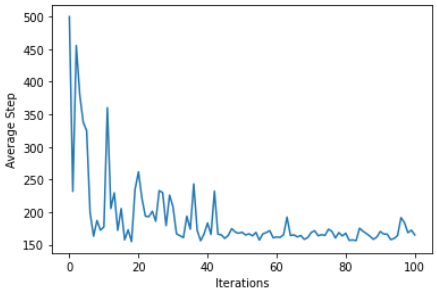

Figure 7: Modified Environment Steps

Our DQN policy can achieve the goal with an average 164.650 steps in 200 consecutive trials.

4.4 Original Environment with Numerical Solution

Zhiqing Xiao [5] provided a numerical algorithm that can solve the MountainCar-v0 without training a model. We modified it to adept TF-Agents. The author's policy is to accelerate to the right if:

$$\min\{-0.09(x+0.25)^2+0.03, 0.3(x+0.9)^4-0.008\} \leq V \leq -0.07(x+0.38)^2+0.07 \tag{5}$$

and accelerate to the left otherwise. This policy can achieve the goal with an average 107.480 steps in 200 consecutive trials.

4.5 Videos

After training each policy, we generated videos of 5 experiments through the method taught in the tutorial. [6] The video names corresponding to different policies are as follows, and they are submitted together with this report.
Random Policy: RandomPolicy.mp4
Original Environment with DQN Policy: OriginalDQNPolicy.mp4
Original Environment with Numerical Policy: NumericalPolicy.mp4
Modified Environment with DQN Policy: ModifiedDQNPolicy.mp4

5 Future Works

Since the birth of TF-Agents in 2018, few people have used it to complete reinforcement learning tasks. To implement the agents on classic control games is a good starting point however still a tip of the iceberg. MountainCar v0 is simple control theory environment in OpenAI Gym, but with discontinuous rewarding function making it hard to train the agent. In future research, we would try to train TF-Agents to solve different knids of games and in addition, try to decompose complex games into parts and then fit speciallized agents one by one to conquer it. We believe the TF-Agents will be a major contributor to more powerful Game AI in the near future.

6 Reference

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Byambaa. Mountaincar v0. https://github.com/openai/gym/wiki/MountainCar-v0.

[3] InstanceLabs. mountaincar.py. https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py.

[4] Ekaterina Gonina et al. Tf-agents. https://github.com/tensorflow/agents.

[5] Zhiqing Xiao. Numerical solution. https://github.com/ZhiqingXiao/OpenAIGymSolution/blob/master/MountainCar-v0_close_form/mountaincar_v0_close_form.ipynb.

[6] TF-Agents Authors. Tf tutorial. https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial.