Creating AI for Mario Kart With Imitation Learning and Reinforcement Learning

Ryo Ikarashi s1280003 Supervised by Prof. Maxim Mozgovoy

Abstract

The objective of this research is to train agents on a racetrack so that they can drive to the goal as fast as possible. In addition to driving along a road, two functions, drifting and items, are also aimed to be used by the agents. In this experiment, imitation learning and reinforcement learning are used to achieve the objective.

1 Introduction

Nowadays, the number of players in esports is more than the one in baseball or soccer. I wanted to develop an AI which can compete with or even outperform humans in the genre of esports games. Mario Kart is one of the most famous racing games, and also used as an esports game. Unlike other racing games, Mario Kart has many functions like items, stage gimmicks, a variety of kart components, etc. In the paper [1], Mario Kart AI was trained. However, they had no drifting capability or powerups. Therefore, I used a Mario Kart-like game with drifting capability and powerups for the environment to work on the experiment.

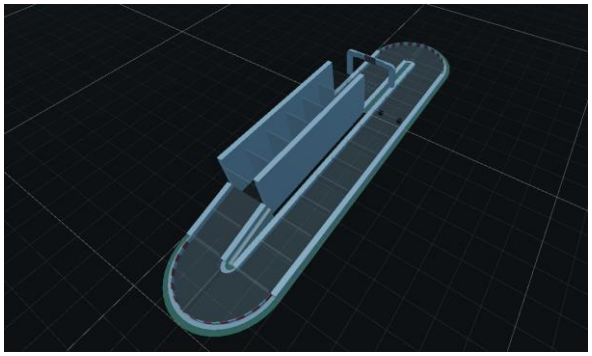
2 Environment

2.1 Tools

Two tools are used to create the environment for the experiment.
The first one is Unity: a cross-platform game engine. This is used to set up the base environment like tracks and kart controlling.
The second one is Unity Machine Learning Agents Toolkit (ML-Agents): an open-source project that enables games and simulations to serve as environments for training intelligent agents.

2.2 Environmental Setup

The Experiment was done in a simple environment.



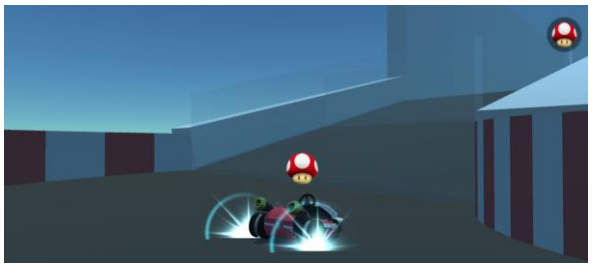


Figure 1 (top). The track used for the training
Figure 2 (bottom). Kart's drifting and holding an item

There is an observation-decision-action-reward cycle when using reinforcement learning.
Observation

2.2.1 Observation

At the observation phase, agents gather the data from the environment. This table shows what kind of data was given to the agents and what was aimed at teaching them with it.

Observation	Purpose
Speed of the kart	Basic move
Rotation of the kart	
Drifting state	Whether it is drifting or not
Accumulated drifting power (the kart gets to boost at certain amount of the power, it accumulates while it is drifting)	It is accumulated while drifting and it gets to boost when reaching certain amount of it

Item state	Whether it has an item or not
Distance between the kart and checkpoints, walls, corner walls and item boxes	What the above states are supposed to be in state with distance of those objects

Table 1. Observation and the purpose of giving them

2.2.2 Decision

At the decision phase, the agent decides the next action based on the data it has. A decision is made every step, which is every 0.1 seconds in the experiment.

2.2.3 Action

At the action phase, the agent makes actions based on its decision. Here are the available actions in my environment. (They are all discrete inputs)

- Moving forward (W key)
- Moving right (D key)
- Moving left (A key)
- Drifting (Space key,
  There are two levels of boosting, and the level is decided by the value of accumulated drifting power. It accumulates while drifting. You can control how much you want to drift inward or outward while drifting with D and A key. Drifting inward accumulates the drifting power faster)
- Using items (U key,
  Only a boost item is implemented in the environment.)

2.2.4 Reward

At the reward phase, the agent gets a reward, which is a numerical signal provided by the environment based on its action. Table1 says what kind of action gives the agent how much amount of reward.

Task	Reward
Going through a right checkpoint	+0.1
Going through a wrong checkpoint	-0.1
Not going through the next checkpoint within a time limit	-0.1
Hitting walls	-0.15
Staying on the wall after hitting it (every 0.02 seconds)	-0.002
Keeping max speed (every 0.02 seconds)	+0.002
Boosting at first level	+0.8
Boosting at second level	+0.9
Getting an item	+0.1
Using an item	+0.1

Table 2. Extrinsic reward (reward defined in the environment)

In addition to these rewards above, -0.0001 reward is given to agents every 0.02 seconds because this small negative reward helps the agent to complete its task as soon as possible. For example, if one episode (period between the beginning of the task and the end of it, in the experiment, reaching the time limit or going through the last checkpoint was set as the end of the task) ended in 1000 steps. Then the agent gets a reward with the amount of 1000 * (-0.001) = -0.1. If one episode ended in 10000 steps. Then the agent gets a reward with the amount of 10000 * (-0.001) = -1.0.

3 Method

3.1 Algorithm

For Imitation Learning, two algorithms are used.

3.1.1 Behavioral Cloning

The first one is Behavioral Cloning (BC), which trains the Agent's neural network to exactly mimic state-action pairs provided in expert demonstrations. There are two types of loss functions (represent how wrong the prediction of the neural network is) of BC. The first one is Cross-entropy Loss, which is used for discreet actions.

$$L(p,q) = -\sum_{i} p_i log(q_i)$$

L is the loss function, p is the true probability distribution and q is the predicted probability distribution.

The second one is the Mean Squared Error (MSE), which is used for continuous actions.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(q_i - p_i)^2$$

MSE is the loss function, n is the number of training examples, p is the true target value and q is the predicted value.

In my experiment, Cross-entropy Loss was used since all of the inputs are discreet.

In BC, agents receive states as inputs directly from expert demonstrations, then try available actions so that it matches the expert state-action pairs.

The advantage of BC is that it doesn't require any environmental interaction during training. Therefore, agents can learn to do complicated tasks faster than using reinforcement learning algorithms. However, its

disadvantages are that because of its exact imitation, it can't generalize its policy well to unseen scenarios, and it lacks in sample efficiency in terms of expert data, which means it requires an expert demonstration with a lot of data.

3.1.2 Generative Adversarial Imitation Learning

The second one is Generative Adversarial Imitation Learning (GAIL). This is the algorithm.

1: Input: Expert trajectories τE ∼ πE, initial policy and discriminator parameters θ0, w0
2: for i = 0, 1, 2, . . . do
3: Sample trajectories τi ∼ πθi
4: Update the discriminator parameters from wi to wi+1 with the gradient
$E_{\tau_i}[\nabla_w log(D_w(s,a))] + E_{\tau_i}[\nabla_w log(1 - D_w(s,a))]$
5: Take a policy step from θi to θi+1, using the TRPO rule with cost function log(Dwi+1 (s, a)).
Specifically, take a KL-constrained natural gradient step with

$$E_{\tau_i}[\nabla_\theta log\pi_\theta(a|s)Q(s,a)] - \lambda\nabla_\theta H(\pi_\theta),$$
$$where\ Q(\overline{s},\overline{a}) = E_{\tau_i}\left[log\left(D_{w_{i+1}}(s,a)\right)\middle|s_0 = \overline{s}, a_0 = \overline{a}\right]$$

6: end for [2]

Here is a quick overview of how it works.

1 Agents take observation and output an action.
2 The discriminator (a second neural network) outputs a reward based on how close it believes the state-action pair provided by the agents is to the one in an expert demonstration.
3 The agent tries to learn how to maximize this reward, which is to try to mimic expert's state-action pairs.
4 The discriminator is trained to better distinguish between expert's state-action pairs and agents' ones.
5 Repeat 1 - 4

The advantages of GAIL are that it is efficient in terms of expert data, which comes in handy especially when the tasks to be recorded takes time to finish or they are so difficult that humans can't perform well stably. Also, it can generalize its policy well to unseen scenarios while it helps to learn complicated tasks faster by trying to imitate expert demonstration. On the other hand, its disadvantage is that it lacks in sample efficiency in terms of environmental interaction.

3.1.3 Proximal Policy Optimization

For Reinforcement Learning, Proximal Policy Optimization (PPO), specifically, PPO-clip is used. This is the objective function of the PPO.

$$L^{CLIP}(\theta) = E_t\left[min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t, clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1 - \varepsilon, 1 + \varepsilon\right)A_t\right)\right]$$

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$
$$where \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \quad [3]$$

L is a loss function, π is a policy, A is an advantage function (represents how good it is to take an action Y at state X compared to other actions according to the current policy), ε represents how much the policy can change, γ represents importance of future rewards, λ is importance of future rewards in advantage calculation, δ represents discrepancy between the current rewards and the expected rewards, and V is a value function (gives the expected cumulative reward in state s using the current policy)

This algorithm prevents a policy from changing too much by "clipping" when the ratio the new policy to the old policy is lower or higher than the set value.

The advantages of using reinforcement learning are that it can generalize its policy well to unseen states and it tries out new actions to discover potentially better strategies. Whereas its disadvantages are that it often requires a lot of interactions with the environment, which makes the training time-consuming. Also, designing appropriate reward functions is challenging since poorly designed ones can lead to undesirable behavior.

3.2 Training Process

Judging from the advantages of each algorithm described before, it is assumed that combining BC, GAIL and PPO can compensate for each other's disadvantages and train neural networks better and faster. The flow of training is described below.

3.2.1 Pre-training

BC was used to train a neural network model. Its policy will be used as an initial policy of a new model which will be trained using GAIL and PPO. The reason for this is that due to no necessity of the environmental interaction in BC, it can help train a neural network which utilizes GAIL, which requires the environmental interaction, at initial steps. However, it has to be noted that training with BC too much can cause creating inflexible model, which gets overly used to the environment used for the training so that it can't perform well in other environments.

3.2.2 Training with the pre-trained policy

A new model is created with the initial policy trained by BC. Using GAIL both and PPO help the neural network learn complicated tasks fast, be flexible to environments, and potentially find better strategies than those of experts. Table 2 shows the configuration used for the experiment.

Configuration	Value
Time Horizon	64
Hidden Units	128
Number of Layers	2
Learning Rate	0.0003
Batch Size	512
Buffer Size	10240
Beta	0.005
Epsilon (ε)	0.2
Lambda (λ)	0.95
Gamma (γ)	0.99
PPO Strength	0 -> 1.0
GAIL Strength	0 -> 0.1
BC Strength	1.0 -> 0

Table 3. Configuration
Time Horizon: How many steps of experience to collect per-agent before adding it to the experience buffer
Batch Size: Number of experiences (a tuple of [Agent observations, actions, rewards] of a single Agent obtained after a Step) in each iteration of gradient descent
Buffer Size: Number of experiences to collect before updating the policy model
Beta: Make its model take more random actions
Strength: How much it can influence the model
[4]

3.2.3 Expected Result

The neural network model trained by BC, GAIL and PPO is expected to learn faster than other combinations of those and be able to work on tracks which was not used for the training, and of course drive along the course using the drifting function and the items properly.

4 Result and Discussion

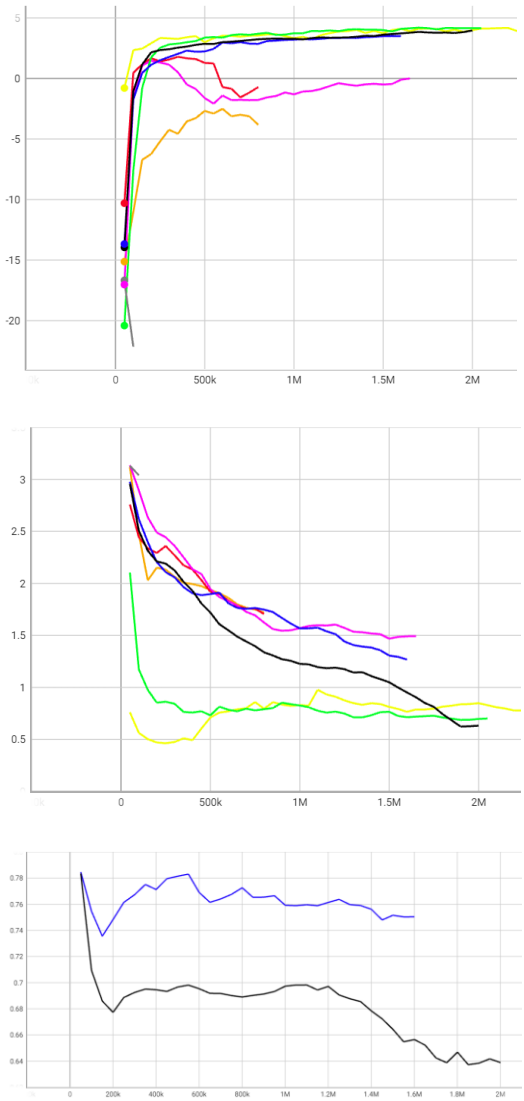


Figure 3 (top). Cumulative Reward
Figure 4 (middle). Entropy (randomness of actions)
Figure 5 (bottom). GAIL Expert Estimate
X-axis: Number of steps
Y-axis: Environment Reward
Yellow: BC + GAIL + PPO (BC 500,000 steps, Demo: 4 times larger steps than others)
Green: BC + GAIL + PPO (BC 500,000 steps)
Black: BC + GAIL + PPO (BC 100,000 steps)
Blue: BC + GAIL + PPO (BC 100,000 steps, GAIL Strength: 10 times smaller than others)
Purple: GAIL + PPO
Red: BC + GAIL (BC 100,000 steps)
Orange: GAIL
Gray: BC

Figure 3 shows that the way of using all three of BC, GAIL and PPO got stable in gaining the possible

highest cumulative reward the fastest. The result of successful training can also be seen in Figure 4. The entropy decreased as time passed, meaning that the agents gradually started taking actions less randomly and got to know which action to take in which state. However, the trained model was not flexible enough to drive on other tracks even though it achieved the goal of driving along the trained track smoothly while drifting and using items properly. In addition, Figure 5 shows that when the GAIL strength is low, it can't learn to take actions like an expert well. While if it is high, it focuses more on gaining intrinsic reward, which is given by the discriminator, rather than gaining extrinsic reward. Therefore, it is difficult to find the way to train the model to imitate experts and also to try to find better strategies than expert's ones. But, in Figure 2, more steps in the expert demonstration, and in the pre-trained model showed that they converged to gaining the highest cumulative reward faster than those with less of them. Thus, giving more data and changing the steps of pre-training are also the options for the improvement besides configuring parameters.

5 Conclusion

Using Imitation Learning and Reinforcement Learning, the neural network was successfully trained to use the drifting function and the items properly in addition to driving along the courses. However, it lacked in the generalized policy, which is required for driving in unseen environments. In order to achieve it, properly setting reward design, observation data and configuration and patience are required.

References

[1] Harrison Ho, Varun Ramesh, Eduardo Torres Montano, "Neuralkart: A real-time mario kart 64 ai", Google Scholar, 2017

[2] Jonathan Ho, Stefano Ermon, "Generative Adversarial Imitation Learning", arXiv:1606.03476 [cs.LG], 2016

[3] John Schulman, "Proximal Policy Optimization Algorithms", arXiv:1707.06347 [cs.LG], 2017

[4] Unity Technologies, "Training Configuration File", https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Training-Configuration-File.md#training-configuration-file, accessed 2024/02/01