Reinforcement learning for obstacle avoidance application in unity ml-agents

Reza Mahmoudi [1] and Armantas Ostreika [2]

[1] *Kaunas Technology University (KTU), Kaunas, Lithuania*
[2] *Kaunas Technology University (KTU), Kaunas, Lithuania*

Abstract

Progress in the field of artificial intelligence has opened up new avenues for researchers to tackle previously challenging use cases. One such example is the automation of simulated autonomous driving, which has long been recognised as a difficult task. However, with advances in reinforcement learning (RL), researchers have been able to achieve satisfactory outcomes. The Proximal Policy Optimisation (PPO) algorithm of RL was used to test models on racecar agents in a unity environment, as described in this paper. The ML agents' framework within Unity Engine is particularly useful for experimenting with RL algorithms. Behaviour cloning is a commonly used technique in the field of machine learning which involves training a model using demonstrations by an expert. This method has been widely employed in various domains such as robotics, autonomous driving, and gaming, and generative adversarial imitation learning, also known as Gail, is a type of Reinforcement Learning technique used to learn policies from demonstration data in situations where the distribution of actions is unknown. Gail utilises a generator and discriminator network that work together to learn a policy that can imitate the behaviour of an expert. Training agents to comprehend their surroundings and overcome obstacles was accomplished by utilising both behaviour cloning and Gail techniques. In the experiment, various obstacles were introduced into the environment and the combination of behavioral cloning as a pre-training technique and Generative Adversarial Imitation Learning (GAIL) were utilized to train for navigating around these obstacles. The optimal model achieved a cumulative reward of -1.619 and a value loss of 0.019 using the aforementioned behaviour cloning method with the use of the PPO algorithm.

Keywords

Reinforcement-Learning, Autonomous driving, ML-Agents

1. Introduction

The technique as Reinforcement Learning (RL) has gained popularity in the realm of game artificial intelligence (AI), as it involves agents learning how to play games by means of repeated experimentation and learning from their mistakes [1]. Furthermore, Reinforcement Learning (RL) techniques are significant in the realm of autonomous driving, as they can facilitate the creation of self-driving systems that are capable of making decisions in intricate and unpredictable environments, including scenarios such as navigating environment or avoiding obstacles on environment [2]. Policy Gradient methods are a group of reinforcement learning algorithms that aim to acquire a policy function that maps states to actions. Proximal Policy Optimization (PPO) is a distinct policy gradient method that incorporates a surrogate objective function that restricts the update step of the policy. This constraint ensures that the updated policy does not deviate excessively from the previous policy, preventing instability issues during the learning process [3]. In this paper, we use ML-agents toolkits that technique utilizes reinforcement learning methodology to aid developers in training their created game through ML implementation. By doing so, the trained model can replicate the entire process, allowing for a comparison of differences. [4] especially the PPO algorithm for training kart agents to

28th Conference on Information Society and University Studies (IVUS'2023), May 12, 2023, Kaunas, Lithuania
EMAIL: reza.mahmoudi@ktu.edu (R. Mahmoudi); armantas.ostreika@ktu.lt (A. Ostreika).
ORCID: 0000-0001-7451-4387 (A. 1); 0000-0001-5718-3766 (A. 2).

navigate the environment and avoid obstacles with Behaviour Cloning that is capable of learning directly from a vast number of human-driven vehicles without the need for a fixed ontology or additional manually labelled data [5] and Gail is frequently employed for imitation learning. This algorithm uses positive demonstrations as a means of mimicking the actions of an expert [6].

2. Materials and Methods

2.1. Reinforcement Learning for Kart Agents

In the context of Reinforcement Learning (RL), an agent learns how to make decisions through interaction with an environment with the aim of maximizing cumulative reward over time. A specific application of this is in autonomous cart racing, where multiple agents, represented by autonomous carts, navigate a track while competing each other at the finish line as quickly as possible. The RL algorithm utilizes the Bellman equation to approximate the value of a state-action pair, denoted as Q(s,a), which represents the expected cumulative reward achieved by taking action a in state s and fallowing the optimal policy subsequently. The Q-value is iteratively updated during the training process according to the given formula [7].

$$Q(s,a) < -Q(s,a) + alpha \times (r + gamma \times \max Q(s'.a') - Q(s.a))$$

2.1.1. RL Sequence Diagram

The sequence diagram in Figure 1 depicts how reinforcement learning agents are given Action A(t) by the environment, receive Reward R(t) from the environment for each action taken, and obtain State S(t) for the current scenario.

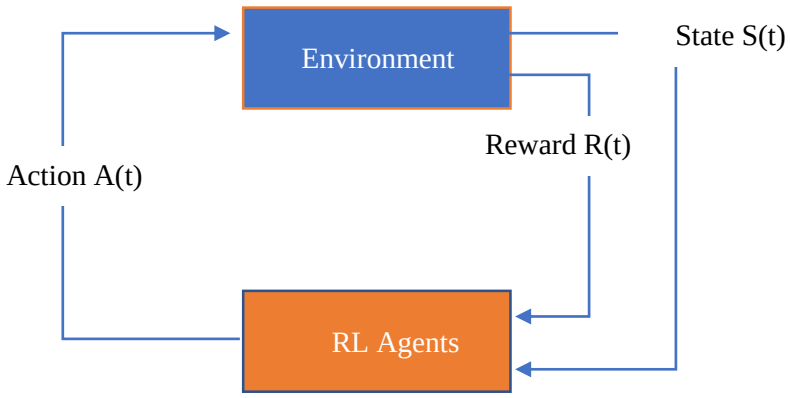


Figure 1: RL Sequence Diagram

2.2. Environment

The simulation was created using the Unity game engine. For the project's experiments, a publicly available environment called "unity-ai-racing-karts-ml-agents" was utilized. This environment comprises 24 car racing tracks and a racing environment. All agents were trained independently. Figure 2 shows the environment.

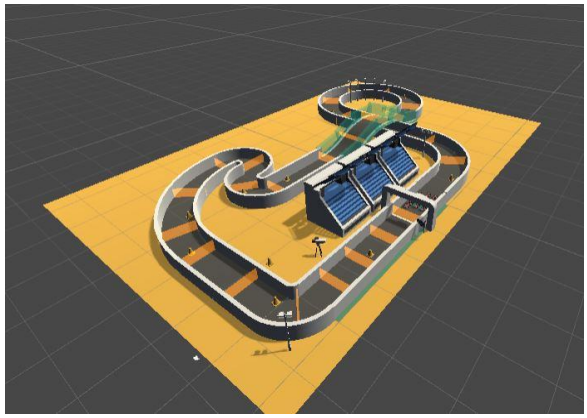


Figure 2: Representation of Unity Environment

In Figure 3, you can observe 24 Kart agents, and using two different methods were utilized to train the agents in our experiments. The first approach involved using ml-agents to control the agents. The second approach was based on "Behavior cloning and Gail" in the behavior type, which utilized the demonstration [8] option available in the Unity game engine.

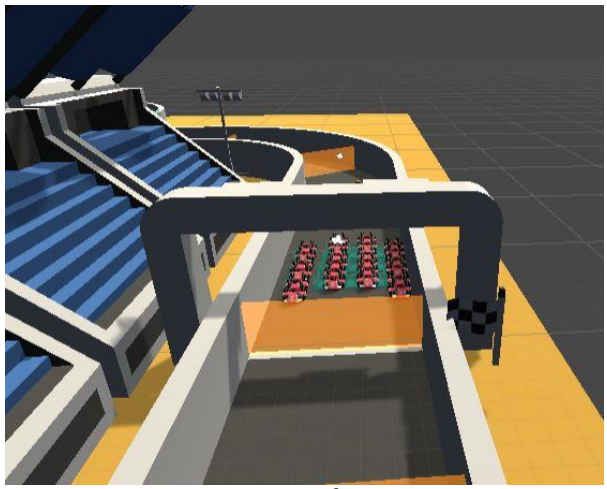


Figure 3: Kart Agents for training

For our experiment, we added random obstacles to train our agents with the demonstration model, which taught them how to navigate the environment and avoid obstacles. The obstacles can be observed in Figure 4 below.

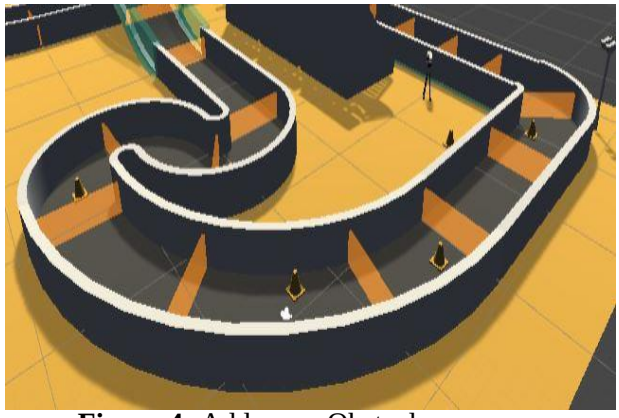


Figure 4: Add some Obstacles

3. Algorithm
3.1. PPO Algorithm

In the ml-agents framework, the PPO algorithm involves several steps such as updating the policy and value networks and using the updated policy to control the agent's behavior in the environment. To ensure the stability and convergence of policy updates, the algorithm uses GAE [9]. Additionally, to avoid large changes that could hinder the learning process, the clipping parameter restricts the size of pf updates. You can see the PPO pseudo code with ML-Agents below.

PPO Pseudo code

PPO with ML-Agents

```
Initialize policy network with random weights θ
Initialize value network with random weights ϕ
Set learning rate α and clipping parameter ϵ
Set number of training iterations T

for t = 1 to T do:
   Collect batch of experiences using current policy πθ
   Compute advantages A using GAE
   Compute target values V using bootstrapped returns
   Train value network to minimize MSE loss between V and
predicted values
   Compute the surrogate objective L_clip using the current
policy πθ and advantages A
   Compute gradients ∇L_clip w.r.t. policy parameters θ
   Clip gradients to avoid large changes
   Update policy using clipped gradients and learning rate α
   Update the environment with the updated policy
end for
```

1. Initialize policy network with random weights θ: A neural network is created with random weights to represent the policy, which is the function that maps observations to actions.

2. Initialize value network with random weights ϕ: Another neural network with random weights is created to represent the value function, which is the function that estimates the expected total reward from a given state.

3. Set learning rate α and clipping parameter ϵ: The learning rate determines how much the model weights are adjusted with each update, and the clipping parameter limits the size of the parameter updates to prevent too much change at once.

4. Set number of training iterations T: This sets the number of times the algorithm will repeat the training process.

5. Collect batch of experiences using current policy $\pi\theta$: The agent interacts with the environment to collect a batch of experiences (state, action, reward, next state) using the current policy $\pi\theta$.

6. Compute advantages A using GAE: The Generalized Advantage Estimation (GAE) method is used to calculate an estimate of the advantage of each action taken by the policy, which reflects how much better the action was than expected.

7. Compute target values V using bootstrapped returns: The bootstrapped return is an estimate of the expected future reward from a given state and is used to compute a target value for each state-action pair.

8. Train value network to minimize MSE loss between V and predicted values: The value network is trained to minimize the Mean Squared Error (MSE) loss between the predicted values and the target values.

9. Compute the surrogate objective L_clip using current policy $\pi\theta$ and advantages A: The surrogate objective is used to estimate how much the policy should be updated, and is computed using the advantages and the current policy $\pi\theta$.

10. Compute gradients ∇L_clip w.r.t. policy parameters θ: The gradients of the surrogate objective with respect to the policy parameters θ are computed using backpropagation.

11. Clip gradients to avoid large changes: The gradients are clipped to limit their size and prevent the policy from changing too much at once.

12. Update policy using clipped gradients and learning rate α: The policy is updated using clipped gradients and the learning rate, which adjusts the policy to improve its performance.

13. Update the environment with the updated policy: The agent interacts with the environment using the updated policy to collect new experiences, and the process is repeated for a set number of iterations.

3.2. Behavior Types

3.2.1. Behavior Cloning

Behavior cloning refers to the process of teaching a neural network to replicate the driving actions of an experienced driver. To achieve this, a dataset of such driving behaviors is used to train the network, which is then used to predict driving actions based on real-time sensor data from the autonomous vehicle. The network is continuously improved over time by gathering new data and refining its training. We have obtained the best result from the behaviour cloning method. In the context of playing a game, observations of the game are represented as s ∈ S and actions as a ∈ A. There is no consideration for time, rewards, or terminal/initial states. Behavioral cloning involves the task of learning the probability distribution p(a|s) of actions taken by human players in a given state s, based on dataset D of tuples (s, a). After learning this distribution, the agent can play the game by selecting an action a ∼ p(a|s) for a given state [10].

3.2.2. Generative Adversarial Imitation Learning (Gail)

Imitation Learning is a technique that focuses on training agents to replicate expert behaviors based on demonstrations. To achieve this, the problem is modelled as a Markov Decision Process (MDP) and a policy π(a|s) is learnt from the state action trajectories τ = (s0, a0, · · · , sT) of the expert behaviour. A more recent approach to imitation learning is Generative Adversarial Imitation Learning (GAIL), which is designed to handle complex, high-dimensional physics-based control tasks. GAIL involves using Generative Adversarial Networks (GANs) to create an adversarial learning framework. The generator network of the GAN represents the agent's policy π, while the discriminator network serves as a local

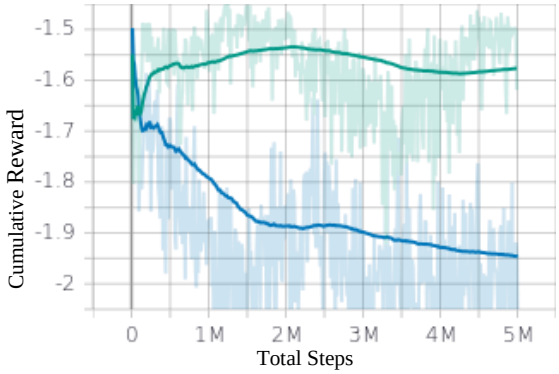
reward function and learns to differentiate between state-action pairs from the expert policy πE and the agent's policy π. This can be expressed mathematically as an optimization problem. [11]

$$\min \pi \max D \; E\pi[\log D(s,a)] + E\pi E[1 - \log D(s,a)] - \; \lambda(\pi)$$

4. Result and Experiment

4.1. Testing with Behavior Cloning and Gail

Using Tensorboard, the experiments involved training agents to navigate obstacles using hyperparameters of the PPO algorithm, and utilizing behavior cloning and the Gail model with demonstration. We have obtained our results by comparing them with those obtained in 5 million steps. In figure 5, we have observed value rewards of -1.619 and -2.092 for Behavior Cloning (green line) and Gail (blue line), respectively. By employing behavioral cloning as a preliminary training phase, the agents were capable of acquiring the intended behavior, and our outcomes were promising in contrast to the Gail approach. Figure 6 shows a loss value of 0.0192. The performance of the agents in the Gail experiment was unsatisfactory since they faced difficulty in navigating the track, failed to evade obstacles, and did not achieve favorable rewards or value losses. Figure 7 illustrates that the Gail value loss had a value of 0.090.



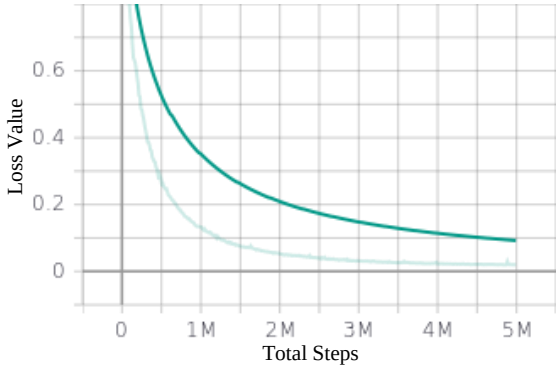


Figure 5: Comparing Reward Value Figure 6: Behavior Cloning Loss Value

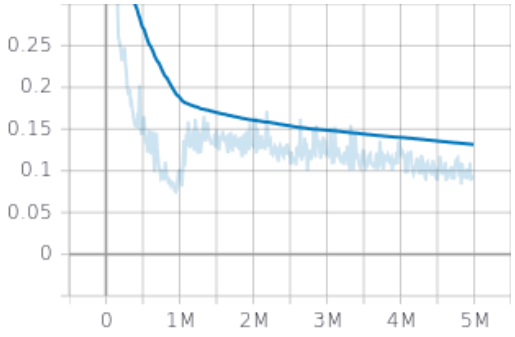


Figure 7: Gail Loss Value

Table 1 displays the utilization of distinct hyperparameters for the behavior cloning and Gail methods, despite employing the same PPO configuration.

Table1: ML-Agent's Hyperparameter

PPO	Behavior Cloning	Gail
batch_size: 120	strength: 5.0	strength: 0.5
buffer_size: 12000	keep_checkpoints: 5	gamma: 0.99
slearning_rate: 0.0003	max_steps: 15000000	network_settings:
beta: 0.01	time_horizon: 1000	normalize: true
epsilon: 0.2	summary_freq: 12000	hidden_units: 128
lambd: 0.95	threaded: true	num_layers: 2
num_epoch: 3		learning_rate: 3e-4
learning_rate_schedule:linear		use_actions: false
network_settings:		use_vail: false
normalize: true		
hidden_units: 256		
num_layers: 2		
vis_encode_type: simple		

5. Conclusion

This research study investigates the application of reinforcement learning (RL) algorithms using the Unity ML-Agents toolkit to train kart agents to navigate a simulated racing track. Various RL algorithms and configurations were compared to assess their performance in training the kart agents to traverse the track successfully and avoid obstacles. The study also identifies the optimal approach for training the kart agents to avoid obstacles on the track with demonstration method especially Behavior Cloning and Gail. In our result, the agents were trained using behavioral cloning beforehand, and they successfully learned the desired behavior with comparing the results with the Gail method. Furthermore, the authors physically inputted a desired behavior and recorded it. The model used behavioral cloning to achieve acceptable outcomes, where the agents successfully avoided obstacles and finished the course. These results were compared to those of the Gail model. The behavior cloning had a reward of -1.619 and a loss of 0.019. The performance of Gail model was not good outcome with -2.092 reward value and 0.090 loss value respectively.

6. References

[1] K. K. .. Volodymyr Mnih, "Playing Atari with Deep Reinforcement Learning," *Arxiv,* 2013.

[2] D. D. T. .. Mariusz Bojarski, "End to End Learning for Self-Driving Cars," *Arxiv,* 2016.

[3] F. W. .. John Schulman, "Proximal Policy Optimization Algorithms," *Arxiv*, 2017.

[4] A. N. &. M. Biswas, Neural Networks in Unity, Springer, 2018.

[5] E. S. .. Felipe Codevilla, "Exploring the Limitations of Behavior Cloning for Autonomous Driving," *ICCV*, 2019.

[6] G. Lee and .. Dohyeong Kim, "MixGAIL: Autonomous Driving Using Demonstrations with Mixed Qualities," *IEEE International Workshop on Intelligent Robots and Systems (IROS)*, 2020.

[7] Z. Ruiming and .. Liu Chengju, "End-to-end Control of Kart Agent with Deep Reinforcement Learning," *IEEE International Conference on Robotics and Biomimetics*, 2018.

[8] V.-P. B. .. Arthur Juliani, "Unity: A General Platform for Intelligent Agents," *Arxiv*, 2018.

[9] P. M. .. John Schulman, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *Arxiv*, 2015.

[10] J. P. V. H. Anssi Kanervisto, "Benchmarking End-to-End Behavioural Cloning on Video Games," *Arxiv*, 2020.

[11] M. S. .. Arjun Sharma, "Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations using Directed Information," *Arxiv*, 2018.