Jet Racing Bot

Fall 2020

Zaiyan Alam
alamm@usc.edu

Hanieh Arabzadehghahyazi
arabzade@usc.edu

Hunny Vankawala
vankawal@usc.edu

Raunaq Porwal
raunaqra@usc.edu

Abstract

The goal of this paper is to summarize our findings while training a game playing agent for our AI Jet-Bot racing game. In this paper, we explain the development of an extremely competitive ML agent (PPO-2), which is our proposed modified version of baseline Deep Reinforcement Learning policy gradient method - Proximal Policy Optimization (PPO) with a Deep Neural Network (DNN) i.e Policy Network along with Curriculum Learning and Long Short Term Memory networks (LSTMs) in order to train our ML agent (aircraft) to fly and dynamically interact with the 3D environment. The training is done through a system of awards and penalties providing the agent with incentives in order to make decisions and complete the navigation in optimal way.

I. Objective

We aim to develop a Human player vs ML Agent 3D racing game. The autonomous jet racing bot is trained as an opponent to a Human player through different world scenarios driven by a physics engine. In order to provide an immersive and competitive experience multiple ML agents with different underlying ML models were used, namely Simple RL model, PPO and PPO-2 (Curriculum and LSTM). We hope to publish our game in future through a WebGL build so that it can provide a fun and challenging game-playing experience to the users.

II. Motivation

Jet-Bot Racing game burgeons heavily from the Human vs AI games genre. Thus, this factor along with the growing Deep Learning methodologies aspect highly motivated us to come up with this challenging game. It works at the intersection of Deep Reinforcement Learning methods like PPO, deep learning methods like LSTMs and the real world physics which is exhibited within the game play. The same is further applied on generalized race tracks with copious amounts of obstacles for the ML agent to dodge. Thus, building a ML agent which displays such adaptability and flexibility given the complexity of tracks, highly motivated us to work on this game.

III. Background

Our Jet-Bot racing game demands multiple ML agents (aircrafts) to compete with each other inorder to win the race. In order to make the game more challenging and involved we have enriched the aircrafts with capacities to enhance their capabilities if they successfully avoid certain obstacles or fly through all checkpoints without any collisions. Apart from conventional flying actions, we have provided the Boost capability to the aircrafts in order to fly 'smart' not just correctly. Integrating the reward based learning not only makes the interaction with the aircraft more intuitive but also helps aircrafts make optimized decisions while navigating.
As shown in the figure below, our Race path and 3D environment is composed of certain buildings and bridges on the green terrain, which the ML agent must avoid colliding with. There are a number of checkpoints which the ML agent must pass through in order to win the race.

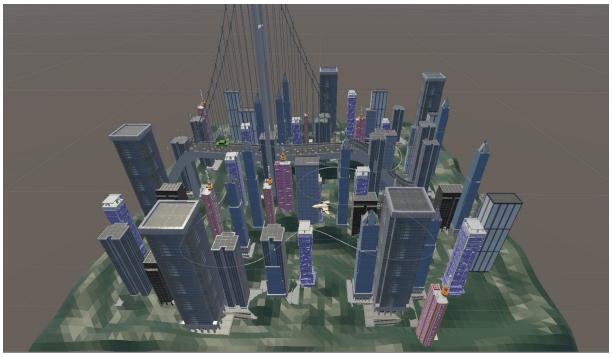

Fig. 1. Race Path

IV. Related Work

The availability of Reinforcement Learning algorithms for training of agents in a dynamic gaming environment is rampant [4,5], with examples like Deep Q-Network (DQN) [7], State Action Reward State Action (SARSA) [6], Soft Actor Critic (SAC) [3], Proximal Policy Optimization (PPO) [10] and many more. The aforementioned algorithms exhibit several strong performance traits and weaker ones also. However, our game environment is a challenging one with continuous observations rather than discrete observations. Thus, SARSA and DQN fall short in this scenario as both are more suitable for discrete environments rather than continuous environments. Moreover, in order to give competitive edge we needed a model that could exhibit Hard level gaming experience. PPO is one the most conventional and advanced Deep RL models but we observed it was somewhat beatable by the human player a lot of times.

Henceforth, we modified the baseline Policy Gradient methods as it is extremely good at handling continuous environment observations and our optimizations through LSTM + Curriculum Learning will improve the performance of our ML agent.

V. Design Approach

The proposed methodology in the paper uses Deep reinforcement learning algorithm in the subdomain of Policy Gradient methods along with a Deep Neural Net i.e Policy Network to train the aircraft agent. In order to give "memory" to our ML agent we have used LSTM along with Curriculum learning. There are certain actions assigned to the aircraft agent like Rotate left or right, Move left or right and Glide up or down. The aircraft agent's objective involves going through each checkpoint without colliding with the obstacles (like buildings, checkpoints and bridges) or the environment (the green terrain).

A. *Proximal Policy Optimization(PPO-2)*

PPO is a sub-domain of Deep Reinforcement Learning methods. Reinforcement learning is form of Markov Decision Process which mainly comprises of the following key components:

- A finite set of states S,
- A distribution of starting states $p(s_0)$.
- We can have a terminal state S_T
- A set of actions A for all the agents
- Transition dynamics (policy) $\pi_\Theta(s_{t+1}|s_t, a_t)$ that map a state/action pair at time t onto a distribution of states at time t+1 using parameter set Θ
- An instantaneous reward function $R(s_t, a_t, s_{t+1})$ associated with each transition.

As mentioned earlier PPO algorithm is a policy gradient method which gives high performance regarding continuous action space and environment. The PPO algorithm uses Policy network, a Deep Neural Network (DNN, as shown in the figure. 2) in order to optimize the objective function by running a number of epochs of Gradient descent over mini batches in order to give optimal Policy as output. Another great advantage of PPO is that it avoids large Policy updates by using a ratio that shows the difference between the Old vs New policy, thus improving performance and training experience.

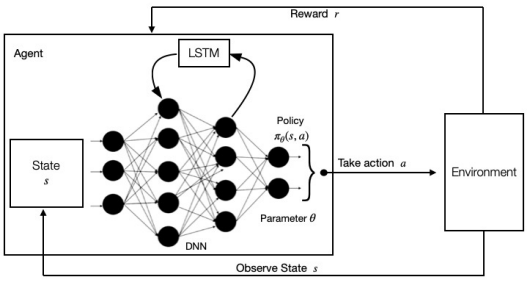

Fig. 2. PPO

$$L^{Clip}(\Theta) = \hat{E}_t[min(\frac{\Pi(a_t|s_t)}{\Pi_{old}(a_t|s_t)})\hat{A}_t, clip(\frac{\Pi(a_t|s_t)}{\Pi_{old}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t] \quad (1)$$

B. *Long Short Term Memory Network*

Reinforcement Learning is a Markov decision process which is memory less and this proves to be an obstacle for our PPO agent which will step back and forth or circles. In order to overcome this obstacle, we included an feedback loop of LSTM recurrent connection between the 2 hidden layers of the Policy network in our PPO-2 model. An LSTM is a gated recurrent neural network capable of learning longer-term dependencies (sequences) making it ideal to provide a memory layer for the agent.

LSTM networks are essentially made up memory blocks called cells which are responsible for remembering and memory manipulations that update the hidden state (memory). As mentioned before LSTM are gated so the cells can store or delete information by opening or closing the gates and that is how LSTMs are able to read, write, delete information from the memory. In our proposed PPO-2 model we have incorporated a sequence of 8 cells, which helps the agent remember past 8 steps of experiences in the decision making process and provide optimal Policy for a certain action. Now our agent must remember how to maneuver when a checkpoint arrives and it remembers what is the optimal way to make 'smart' turns which helps our agent to take the lead. Since LSTMs are recurrent, back propagating the output errors helps the network to learn over time steps.

C. *Curriculum Learning*

Deep Neural Networks (DNNs) can be trained speedily by using the information learned from past tasks. To achieve that, there are several techniques such as transfer learning, multitask learning and curriculum learning. Here we dive deep into curriculum learning but before learning more about curriculum learning, let's first learn about curriculum. Curriculum is basically a series of lessons which is essentially training criteria and curriculum learning initially starts with a very basic task and eventually increases the complexity as learning progresses until we are able to develop the model we need. In the process, it remembers previous learning instances. During each lesson, a set of weights is generated and it is built upon the previous weights. We initially begin with the size of the diameter of checkpoints as 55m and eventually keep decreasing the diameter at each lesson by 5m, so by lesson 4 we have checkpoints diameter to be 40m. For cumulative_reward gain of +3 the lesson is switched to another and diameter is decreased. At the end of the sequence, our agent can efficiently maneuver through the checkpoints as it has not forgotten the knowledge gained during the first lesson so it can efficiently navigate through checkpoints of smaller diameters from the knowledge gained during the final lesson.

D. *States*

In our gaming environment, a particular state is defined through a collection of observations : velocity, vector distance

to the next checkpoint and orientation of the next checkpoint. Since the observations are made over a 3D environment, a total of 9 observations are made. These observations are saved into a vector format of 9 float values, at every 5 timesteps (equivalent to 0.1 seconds) and then fed into the neural network. Raycasts (over 300m long) are also used in order to make observations regarding the objects around the ml agent which helps in object detection and collision avoidance.

E. ***Actions***

There are mainly 3 actions used by the agent with positive and negative values of each : Rotate left or right, Move left or right and Glide up or down. Thus the action sequence vector consists of 6 values corresponding to each direction. The agent is also provided with the ability to use Boost in a certain direction with higher speed. If the agent collides with either of the buildings or terrain or goes out of boundary it explodes and respawns on the last visited checkpoint.

F. ***Rewards***

The following reward system is used for training the agent:

(i) Small positive reward for passing through a checkpoint -

$$R(Checkpoint) = 0.5 \tag{2}$$

(ii) Negative reward regarding collision with solid objects -

$$R(Collision) = -1 \tag{3}$$

(iii) Small negative reward if the agent stays still and takes no action -

$$R(Stationary) = -1/(Max\_step\_value = 5000) \tag{4}$$

(iv) Negative reward if next checkpoint not reached within 300 timesteps -

$$R(Checkpoint\_timeout) = -0.5 \tag{5}$$

Therefore, Total cumulative reward :

$$R(Cumulative) = R(Checkpoint) + R(Collision)$$

$$+R(Stationary) + R(Checkpoint\_timeout) \tag{6}$$

G. ***Results***

We used certain hyperparameters from Unity-ML agents config file in order to tune the PPO-2 model performance and achieve better results. We observed the training results on various hyperparameter values regarding the following ones : *batch_size = 512, num_hidden_units = 128, num_hidden_layers = 2, buffer_size = 4*batch_size* and *num_epochs = 6*. In order to tune the LSTM network, we observed optimum performance for hyperparameter values : *sequence_length = 8* and *memory_size = 128*. Regarding the Curriculum learning process, *num_lessons = 4* and *minimum_episode_length = 100* with lessons switching every time a total cumulative reward gain of +3 is achieved. These parameters were chosen to provide a balance between the performance and model complexity so as to avoid poor performance in case the model becomes too complex.

From figure 3 we can compare the baseline PPO model and our proposed PPO-2 model. It's evident from the plot that our PPO-2 model performs better than the PPO as it attains highest *cumulative_rewards = 18.2* much faster, i.e within 2 Million timesteps as compared to 3 Million timesteps used by PPO. This higher performance by our PPO-2 model arises because of the memory power through LSTMs and optimized training through Curriculum Learning.

Fig. 3. Cumulative Rewards per Episode

From figure 4, we can see that the learning rate of the agent decreases linearly with the timesteps which signifies that the agent has learned most of the environment and exhausted most of the observations. One major factor to be considered over here is the fact that the environment is simple, deterministic and static.

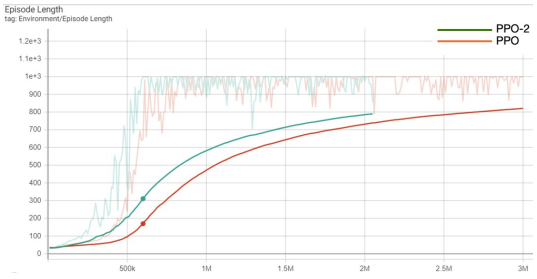

Fig. 4. Episode Length per Iteration

VI. Conclusion

In our paper, we proposed a better model PPO-2 as compared to PPO, which can be used to give a Hard level gaming experience to the Human player. Moreover, in our experiment, we carried out competitive ML agents (PPO-2, PPO and Simple RL model) vs Human races where in around 80% of the cases PPO-2 won. Thus, we have shown that using multiple ML agents in a game with different underlying ML models gives a unique challenging experience. In future, work on experimenting how the behaviour of other advanced Deep RL methods in the game can be observed.

REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

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

[3] Arthur Juliani et al. "Unity: A general platform for intelligent agents". In: arXiv preprint arXiv:1809.02627 (2018).

[4] Kai Arulkumaran et al. "Deep reinforcement learning: A brief survey". In: IEEE Signal Processing Magazine 34.6 (2017), pp. 26–38.

[5] Yuxi Li. "Deep reinforcement learning: An overview". In: arXiv preprint arXiv:1701.07274 (2017).

[6] Gavin A Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[7] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: Nature 518.7540 (2015), pp. 529–533.

[8] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: arXiv preprint arXiv:1509.02971 (2015).

[9] John Schulman et al. "Trust region policy optimization". In: International conference on machine learning. 2015, pp. 1889–1897.

[10] John Schulman et al. "Proximal policy optimization algorithms". In: arXiv preprint arXiv:1707.06347 (2017).

[11] Yoshua Bengio et al. "Curriculum learning". In: Proceedings of the 26th annual international conference on machine learning. 2009, pp. 41–48.

[12] Tambet Matiisen et al. "Teacher-student curriculum learning". In: IEEE transactions on neural networks and learning systems (2019).