Tutorial Report

Caleb Jordening
Occidental College

1 Introduction

2 Methods

2.1 Packages and Plugins and Software, Oh My!

Pip installing the required packages, setting up BakkesMod, having Rocket League downloaded, and installing RLBot were not included in this tutorial; however, it is necessary to do so before jumping into the base code. More details about versions, installations, and how to run a trained model in a game setting are explained further in the methods and software documentation.

2.2 Base Code Setup

To begin the tutorial, a base code was used from an example setup to start training a deep reinforcement learning agent created by programmers for RLGym [1]. This code uses the rlgym-tools library created by RLGym, which provides support for a variety of applications in RLGym; and, simplifies some of the complexities of implementing Stable-Baselines3 in RLGym to train the agent. The raw contents of this code were copied and pasted into a blank python file labeled "tutorial-bot.py". The first variable explained is the frame-skip variable. This variable is in integer value that defines how long an action will be performed until a new action is performed. This value is defaulted and kept at 8. As the frame-skip value approaches 0, the time it takes to train the bot will tremendously increase. The half-life-seconds variable is also not changed, and is defaulted to 5. This value is used to calculate the gamma value for discounting future rewards.

2.3 Explanations and Changes of the Get-Match() Function

Moving forward, the bot that will be trained will learn to play 1v1. So, under the get-match() function, the team-size value is adjusted to 1. Currently, there is a single reward-function that is set to the VelocityPlayerToBall() reward, which is a simple reward function that is apart of the rlgym-tools library. At this point in the tutorial the reward function remains at default; however, it will be changed later on in the tutorial. The self-play variable is set to true. This allows for the agent to play in a 1v1 match against itself — which allows for more fruitful training data and faster learning. If this condition is set to false, the bot will train against "All-Star" a bot created by Psyonix, which is notoriously known for being extremely low-level in terms of skill/ability. Given this information, the data gained from playing All-Star will not be as beneficial and will increase the time it takes to see typical in-game behaviors emerge. That is of course, if they even emerge at all. The terminal-conditions object is set up so that it will reset the game to the default kickoff state if either a goal is scored, or if the timeout condition is met. This timeout condition is beneficial for getting agents out of positions where they are exhibiting non-beneficial, repetitive behaviors like driving in a circle. The obs-builder will be left to advanced obs, which is a observation builder that is provided by rlgym-tools. It is important to note, as the comment in the code suggests, the advanced observation default is "not that advanced" and is a "good default" for starting to build an agent. The tutorial suggests that more complex, custom observation builders can be coded that can provide more useful data about the environment that will be passed to the agent at each new state. Creating custom observation builders is an area in which the overall bot performance and training could be improved. The state-setter is set to default, which will reset the agents and ball back to kickoff position. Finally, the action parser is set to discrete, which limits the amount of actions the agents can take. This is opposed to continuous where there would be an infinite amount of actions to take.

2.4 Creating Useful Variables

After discussing how the Get-Match function objects affect the environment and are used in training, important variables were created to further setup the training. First the amount of steps trained per epoch are defined at 100,000. Next, the number of agents per match was defined. This will vary based upon whether a 1v1, 2v2, or 3v3 agent will be trained. This example is training a 1v1 agent, so agents-per-match is set to 2. Next the number of instances of Rocket League training will be defined. The Epic Games version allows for multiple Rocket League instances to be run simultaneously. Rocket League is not an extremely intense game to run, so having multiple instances where the

agent is training against itself can help speed up the process. However, open too many, and performance could be hindered due to a lack of computational power. The tutorial suggests 20 instances for a 1v1 bot, and 10 for a 2v2 bot. Given that the tutorial did not state their hardware specs, I opted to be conservative and train 5 instances for 12 hours to see what kind of performance that would get me. This num-instances variable was passed through the SB3 multiple instance environment function. Finally, steps and batch-size were variables created that would be used in tuning the hyperparameters.

2.5 Tuning the Hyperparameters

This portion of the tutorial was relatively lackluster and did not provide much information as to why the values were set to what they were. The only explanation is that they were "what works" based off of prior experience. There is certainly room for exploration and research as to how tuning the hyperparameters could affect the model's performance.

2.6 Save Path

Every so often, we want to make a save of what the model has learned from the environment. Hence, within the working directory in which the agent.py is located, three other directories were made. These directories will be titled "logs," "models," and "mmr-models." Hence, the idea is that the TensorBoard logs would be sent to the logs folder, versions of the trained model would be saved to the models folder every specified number of steps or when the program is exited, and a comparison model would be sent to the mmr models directory every specified number of steps. In this example, by using a while loop, the model is saved every 5,000,000 time steps, or whenever the program is closed out and the comparison model is saved every 25,000,000 steps. This save path would then be used to access the most recently trained model. A try and except is implemented to see if there is a saved model readily available to be trained. If not, then a blank model would be created to start training.

2.7 Changing the Reward Function

The base code includes a simple reward function of giving reward to the agent for moving at speed towards the ball. To make this slightly more complex, more reward functions are imported from the RLGym library. In addition to the velocity of the player towards ball reward, the velocity of the ball towards the goal and event rewards are passed through a combined reward function. The former gives a reward for getting the ball to move towards the goal at speed. The latter allows the programmer to set reward values for in game events such as scoring a goal, saving a goal, or getting scored on. Finally, weights were set that are multiplied by the rewards before being returned. This allows for control of how influential each reward will be.

2.8 Modifying the Neural Network Architecture

Finally, the last changes made to the code was creating a modified neural net. The model used a Tanh activation function. The model was comprised of 4 layers. The first two layers were set to have 512 nodes. Then, the network branches into 2 more layers: one that deals with the policy and actions the model is going to take, and the second layer is the value function layer that determines the potential value for a given state-action pair.

2.9 Training the Agent

Next, the code was run and the 5 gym environments were created. Prior to opening all of the instances of Rocket League, it is important to have BakkesMod running with the RLGym plugin enabled. This was done by opening a separate instance of Rocket League, going into the BakkesMod plugins section, and enabling RLGym. While training, the in-game graphics and resolution settings were altered. All graphics settings were set to high performance/performance, all the graphic effects were turned off, the fps was set to "uncapped," and the resolution was set to 640x480 in windowed mode.

2.10 Getting the Bot into RLBot for Evaluation

Next, a wrapper was created to upload the trained models into a 1v1 match via RLBot. This was done by forking and modifying an example bot from the official RLGym/RLGymExampleBot repository [2]. After downloading the contents of the repository, the agent.py file was modified. Under agent file, the act(self,state) method was modified. It was changed to use the best predicted action of the model and return what that action was. Finally, the trained model zip file was put in the directory where all of the contents of the downloaded repo were. The self.actor instance was then modified to locate the zip file of the trained model. This is how the wrapper would load the agent into the RLBot environment.

Next, the bot.py file was modified. First, the default-obs import was replaced with the advanced-obs import to match the observation builders used to train the model. Moving along, there was a need to create a method that would reshape the environment if there was a discrepancy between the number of opponents/teammates the model was trained with and the game setting the bot is being placed in. For example, there is extra information the model has from training to play 3v3, so if the bot is then put in an environment

where it is playing a 1v1, there is extra information that must be dealt with. This bot was trained to play 1v1, so the reshape method was not implemented.

Finally, RLBot was opened and the most recent exit model was uploaded to RLBot. This was done by uploading the bot.cfg file that was in the directory with agent.py and bot.py to RLBot. The agent was trained

3 Evaluation Metrics

The bot was trained on 5 instances for 12 hours. The most recent exit model was evaluated in a 1v1 game scenario against itself and against "Beast from the East," which was considered by RLBot's website as a "B-tier bot." Originally, the plan was to additionally evaluate against an A and S tier bot; however, for reasons we'll get to, this did not happen. Evaluation would occur both qualitatively and quantitatively. Game play would be observed for intelligent behaviors. This could be chasing after the ball, hitting the ball, rotating from offense to defense and vice versa, etc. The bot would also be judged off of whether or not it won or lost the game. Finally, The actual point breakdown would be assessed to see how many saves, epic saves, shots on goal, goals, and ball touches were made.

4 Results and Discussion

Given that the tutorial had 20 instances of Rocket League to train a 1v1 model and that this training had been happening for multiple days, I was not expecting much from my bot that trained for 12 hours with 5 instances. In the first game against itself, both bots ended up doing the same thing. They would drive forward, boost, reverse right before touching the ball, and then get stuck in a loop where they would drive back and forth very quickly. Neither of the bots were able to touch the ball. The game was reset 5 times, and 4 out of the 5 times these same behaviors occurred. One time, however, one of the bots started to reverse so far, that it climbed up the wall and onto the ceiling. After doing so it fell to the ground and then got stuck in the back and forth loop.

Next, the bot was pitted against "Beast from the East." Our trained agent exhibited the same behaviors of accelerating forward, boosting, and the reversing right before hitting the ball. This made it extremely easy for Beast from the East to score off kickoff. This occurred on every kickoff except for one. In the kickoff where the two bots were placed in a straight line to the ball off of kickoff, our model would actually drive towards the ball and block the Beast's shots off kickoff. This happened a couple times; however, it was still a blowout. The final score was 40-0.

The tutorial's bot, after being trained for a considerable amount of time on multiple instances, is supposedly supposed to train a bot that is "rookie level." The rookie level bot is a reference to Rocket League's own bots that they provide. Rookie is the worst out of all 3 bots that Rocket League provides. Certainly, there is room for improvement within the realm of writing custom rewards and reward shaping, writing custom observation builders to pass on more specific data to the agent, and optimizing the neural network's architecture and policy gradient optimization algorithms.

References

[1] RLGym. *rlgym-tools*. 2022. URL: https://github.com/RLGym/rlgym-tools/blob/810d14d4905cdfd525d5b84ed0e2fd0cdc62366a/rlgym_tools/examples/sb3_multi_example.py.

[2] RLGym. *rlgym-tools*. 2022.