ImitationBot

Frank Cline

May 3, 2019

A Rocket League bot trained from the gameplay of better bots.

Figure 1: Rocket League Coverart

1 Introduction

1.1 Rocket League

Rocket League is a video game developed and published by Psyonix[1]. The game plays like soccer with RC cars. Players control an RC car in an arena and compete on a team against one other team. The teams consist of one, two, or three players, and each team has the same number of players. There are two goal posts opposite of each other and a large ball that players can move by colliding with it.

The goal of the game is to move the ball into the opponent's goal. However, the game contains additional complexities. Players can jump their cars into the air with a press of a button, and they can acquire boost to effectively fly for a short duration. Such complexities make it a hard game to train bots, or agents, to play the game. The

goal of this project is to create a bot that can play Rocket League by learning from expert bots through inverse reinforcement learning.

1.2 Training an RL Bot with IRL

The idea of learning based off of some reward function that determines an agent's actions has been explored at least since Littman and Michael released their paper "Markov Games as a Framework for Multi-Agent Reinforcement Learning". This idea explores the notion of training an agent based off of a Markov Decision Process which is defined by a set of states S and a collection of actions A. It also contains a transition function T which defines a probability distribution of the next state given the current state s and action a. There is also a reward function R which returns an evaluation based off of s and a. Lastly, there is the discount factor γ which is in [0,1] and determines how much future rewards should be discounted[2]. In reinforcement learning, the agent looks to maximize their rewards to find an optimal policy that maps states to actions.

Inverse reinforcement learning (IRL) explores the idea that given an expert's actions, the agent can determine the best reward function given the expert's trajectories, i.e., the expert's behavior. IRL harnesses the knowledge of an expert to learn from them. A reward function is determined by the actions of the expert. This allows the agent to learn from an expert. This idea of learning from an expert can be applied to difficult problems such as 3D games with multiple actions. This can be especially applicable to a complex game such as Rocket League. By learning from an expert, agents can determine expert-optimal reward functions to the point of possibly exceeding expert performance.

1.3 Training an RL Bot with an ANN

A simple feed-forward artificial neural network (ANN) can be used to train a bot as well. A feed-forward ANN is composed of a collection of nodes that form layers[3]. Each layer connects to the next layer to form a complete bipartite graph. Input enters the input layer, and it is fed through the neural network to produce the output layer. The outputs for a Rocket League bot would be "throttle, steer, pitch, yaw, roll, jump, boost, brake." All of the values either fit between $[0, 1]$ or $[-1, 1]$. The input would be observations of the environment by the Rocket League bot.

2 Related Work

2.1 Rocket League Bots

There already exists a framework for creating bots for Rocket League call RLBot[4]. This framework allows for the creation and testing of Rocket League bots in offline mode. It provides all of the in-game information necessary for creating a bot. There already exists some bots trained through reinforcement learning such as Saltie which was trained with deep reinforcement learning[5]. However, there are no bots currently available that have been trained through inverse reinforcement learning.

2.2 Inverse Reinforcement Learning

In 2000, Russell and Ng released one of the founding papers on IRL. They explore some rudimentary IRL algorithms, and they also state one of the problems of IRL: degeneracy. Degeneracy is the existence of a large set of reward functions that fit the expert's trajectories[6]. This is a problem because a sub-optimal reward function could be used by an agent. In "A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress" by Arora and Doshi, solutions to degeneracy are explored. One solution is to use a heuristic that favors the learned values of the agent that are closest to the expert's values[7]. Another more popular solution was introduced by Ziebart et. al. in "Maximum Entropy Inverse Reinforcement Learning".

According to Wikipedia, "the principle of maximum entropy states that the probability distribution which best represents the current state of knowledge is the one with largest entropy"[8]. Ziebart et. al. utilize this principle by choosing the distribution with the most entropy that also fits the expert's policy[9]. In 2015, Wulfmeier, Ondruska, and Posner extended Ziebart et. al.'s work in their paper "Maximum Entropy Deep Inverse Reinforcement Learning". They use a convolutional neural network (CNN), a deep neural network that requires little preprocessing generally, to approximate a reward function[10, 11]. They found that using a CNN can work well even in cases where there is a large state space, and a complex reward function needs to be determined[11].

2.3 Other IRL Methods

A large state space can be avoided though. In "Inverse Reinforcement Learning for Video Games" by Tucker, Gleave, and Russell, the authors propose an autoencoder that encodes the input environment of a video game into a smaller space[12]. They found that encoding the state space greatly increased the efficiency of expert samples. In "Imitation Learning with Concurrent Actions in 3D Games" by Harmer et. al., they describe a novel idea of allowing multiple actions to be selected at every time step instead of a single action[13]. This could be especially helpful in the case of Rocket League because there are almost always multiple actions being selected at each time step. Another interesting complexity with Rocket League is the fact that the agents on the same team must work together. In "Multi-Agent Inverse Reinforcement Learning" by Natarajan et. al., they introduce a novel idea where a centralized controller learns to coordinate the behavior of the agents by optimizing a weighted sum of rewards from all of the agents[14]. This could be interesting for Rocket League because most bots don't always work optimally with their teammates.

2.4 Artificial Neural Networks

Python machine learning libraries like Tensorflow[15], Scikit-learn[16], and Keras[17] exist for easily creating artificial neural networks and training them. The combination of these three libraries makes it easy to create and train models. The models require a specified shape like input size and output size, and they also require an optimizer and loss function is specified as well. The optimizer updates the model during training to

Algorithm 1 Maximum Entropy Deep IRL

Input: $\mu_D^a, f, S, A, T, \gamma$

Output: optimal weights θ^*

$\theta^1 = \text{initialise\_weights}()$

Iterative model refinement

for n = 1 : N **do**

  $r^n = \text{nn\_forward}(f, \theta^n)$

  Solution of MDP with current reward

  $\pi^n = \text{approx\_value\_iteration}(r^n, S, A, T, \gamma)$

  $\mathbb{E}[\mu^n] = \text{propagate\_policy}(\pi^n, S, A, T)$

  Determine Maximum Entropy loss and gradients

  $\mathcal{L}_D^n = \log(\pi^n) \times \mu_D^a$

  $\frac{\partial \mathcal{L}_D^n}{\partial r^n} = \mu_D - \mathbb{E}[\mu^n]$

  Compute network gradients

  $\frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n} = \text{nn\_backprop}(f, \theta^n, \frac{\partial \mathcal{L}_D^n}{\partial r^n})$

  $\theta^{n+1} = \text{update\_weights}(\theta^n, \frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n})$

end for

Figure 2: Maximum Entropy Deep IRL Algorithm[11]

approximate the correct values. The loss function is used to measure how good the model's predictions are. Mean squared error was chosen as the loss function for the models. Keras offers numerous optimizers, but three are chosen to optimize three different models. The three optimizers are Stochastic Gradient Descent (SGD), Adadelta, and Adam. All three optimizers use gradient descent which is, "a first-order iterative optimization algorithm for finding the minimum of a function"[18]. SGD has a set learning rate, and it updates the weights of the model for each training sample. Adadelta and Adam are similar to SGD, but they have adaptive learning rates. Adam is just slightly different from Adadelta; it uses a bias correction, and it adds momentum[19].

3 Method

3.1 Collect Data

The RLBot creators have run tournaments in the past, and the standings for the bots are available. In addition, a lot of the top bots are available on GitHub[20]. ReliefBot was ranked as the number three best bot, and it was available on GitHub[21].

Thus, data was collected by having 4 ReliefBots play against each other in 2v2 games on an AWS EC2 instance. Every second, each bot logged the game state relative to themselves and their output. This made training data X and Y. The logs were then uploaded to AWS S3 to be retrieved on a host machine. On the host machine, the logs were converted to CSV files to be used for training. Over 2 million states were collected with 72 features and 8 labels each.

3.2 Create Models

Rocket League is a complex game with a large state space, so maximum entropy deep inverse reinforcement learning (MEDIRL) lends itself nicely to solving this problem. However, the implementation provided by Alger is too complex to use. Thus, three different ANN models are used. The only difference between the three models are the optimizers used by the models: SGD, Adadelta, and Adam. The models have 16 input neurons and 8 output neurons. The layers of each model is as follows $16-32-16-8$. The input is the bot's team, location, velocity, and rotation, and the ball's location and velocity. All of the inputs were normalized between $[-1, 1]$. The outputs are "throttle, steer, pitch, yaw, roll, jump, boost, brake" which fit between $[0, 1]$ or $[-1, 1]$. The perceptrons use the $tanh$ activation function to keep the values between $[-1, 1]$.

3.3 Train Models

The models were trained in batch sizes of 128 over 5 epochs on $800,000$ of the $2,000,000$ states collected. All of the states could have been used, but there were some obvious errors with some of the data. Thus, $800,000$ good data points were used to train the models.

3.4 Test the Models

The model is loaded by the RLBot framework into a Rocket League bot. Performance of the agent is measured by an in-game score assigned to each player. The score relates to how well each player has performed in-game, e.g., making goals, saving goals, and assisting scorers increase a player's overall in-game score for that game. While not perfect, the in-game score will provide a meaningful assessment of the IRL agent's performance.

PEAS	IRL with Rocket League
Performance Measurement	In-game score and win/loss ratio
Environment	Rocket League arena
Actions	Accelerate, steer, pitch, yaw, roll, brake, jump and boost
Sensors	The position, direction, and speed of the ball and other cars

Table 1: The PEAS description for Rocket League IRL.

4 Results

All three models were able to achieve a semi-decent accuracy of 85% with low losses around 0.15. However, even though the accuracy is decent, it still was not good enough. All three models can barely hit the ball, and when they do hit the ball, it's generally an accident. The models learned some interesting behavior though. When the agents start to go for the ball initially, they do a sub-optimal jump, and that is generally the only time the agents jump. Theoretically, the agents should be jumping more often than that. The agents definitely attempt intelligent behavior though. They would often drive close to the ball and just barely miss the ball. That could be due to the lack of accuracy.

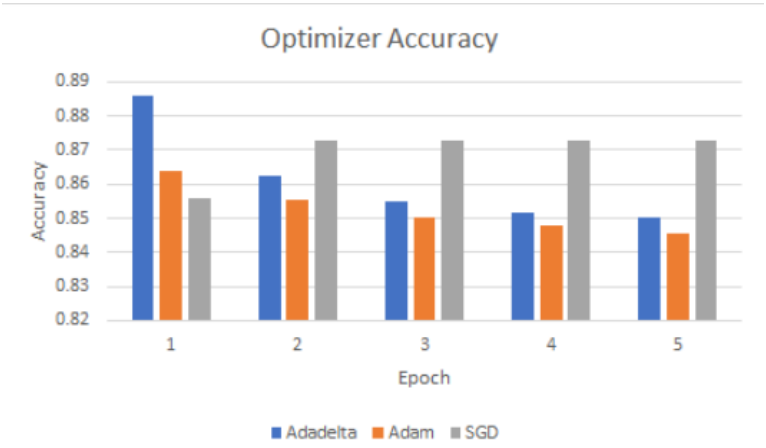

Figure 3: Model Accuracy

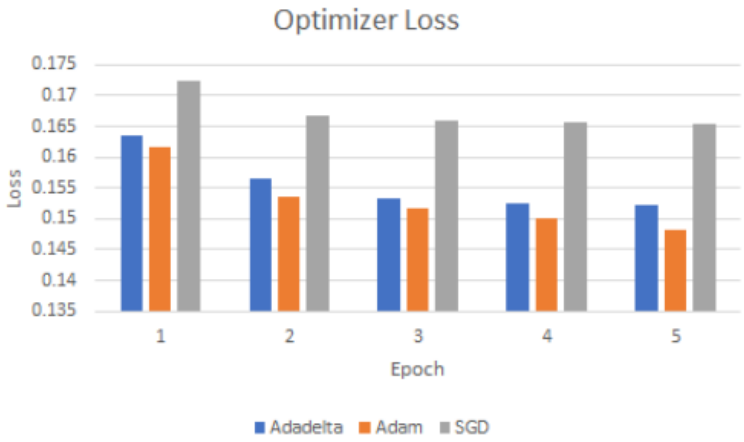

Figure 4: Model Loss[11]

5 Conclusion and Future Work

Rocket League is a complex game. The complexity of Rocket League makes it hard to utilize machine learning to train bots. Bots can be trained using simple feed-forward artificial neural networks; however, given the results of the models used in this paper, other training techniques would probably be better. A variety of improvements could be made to these models. First, more meaningful features could be added. Additional features like the direction to the ball or the speed of the car would be good. Second, principal component analysis could be applied. There exists 72 different features, so it would be beneficial to us principal component analysis to find the most valuable features. Third, it could be better if different bots other than ReliefBot were analyzed. This might make generalization better, but it could also hurt performance given the different play-styles. Lastly, it would be nice to actually apply maximum entropy deep inverse reinforcement learning. A simple ANN couldn't figure out the complexities of Rocket League. However, MEDIRL might stand a better chance.

References

[1] W. contributors, "Rocket league — Wikipedia, the free encyclopedia," 2019.

[2] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*, pp. 157–163, Elsevier, 1994.

[3] Wikipedia contributors, "Feedforward neural network — Wikipedia, the free encyclopedia," 2019. [Online; accessed 3-May-2019].

[4] "Rlbot: http://www.rlbot.org/."

[5] "Saltie: https://github.com/saltierl/saltie."

[6] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, p. 2, 2000.

[7] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," 2018.

[8] W. contributors, "Principle of maximum entropy — Wikipedia, the free encyclopedia," 2019.

[9] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[10] W. contributors, "Convolutional neural network — Wikipedia, the free encyclopedia," 2019.

[11] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," 2015.

[12] A. Tucker, A. Gleave, and S. Russell, "Inverse reinforcement learning for video games," 2018.

[13] J. Harmer, L. Gisslén, J. del Val, H. Holst, J. Bergdahl, T. Olsson, K. Sjöö, and M. Nordin, "Imitation learning with concurrent actions in 3d games," 2018.

[14] S. Natarajan, G. Kunapuli, K. Judah, P. Tadepalli, K. Kersting, and J. Shavlik, "Multi-agent inverse reinforcement learning," in *2010 Ninth International Conference on Machine Learning and Applications*, pp. 395–400, IEEE, 2010.

[15] Wikipedia contributors, "Tensorflow — Wikipedia, the free encyclopedia," 2019. [Online; accessed 3-May-2019].

[16] Wikipedia contributors, "Scikit-learn — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Scikit-learn&oldid=888470387`, 2019. [Online; accessed 3-May-2019].

[17] Wikipedia contributors, "Keras — Wikipedia, the free encyclopedia," 2019. [Online; accessed 3-May-2019].

[18] Wikipedia contributors, "Gradient descent — Wikipedia, the free encyclopedia," 2019. [Online; accessed 4-May-2019].

[19] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[20] "Rlbot player list: https://braacket.com/league/rlbot/player."

[21] Tarehart, "Reliefbot," Mar 2019.