Capstone Proposal

Ben Thayer

March 6th, 2018

Domain Background

Reinforcement learning and control theory have been working for many years to solve the problem of controlling robots. The general problem is mainly concerned with an actor in an environment that has the ability to choose actions to change the state of the environment in order to receive a reward. This is a field that has been researched for many years and is still being improved on today. There is a framework called Gym made by OpenAI that aims to present these problems in a consistent manner. Both classical techniques and modern techniques can be used to solve the problems offered within Gym “environments.” One of the advancements that has been made recently is deep Q-learning. It is a technique that allows for use of visual input and other types of more complicated inputs in trying to choose actions. One well-researched use case of this is in Atari games. Q-learning is able to take in the screen as input and the score as a reward signal and outputs actions which then lead the model to score higher than any human.

I am particularly interested in this area of research because I believe that reinforcement learning can tackle problems that humans will never be able to. I also love the intellectual challenge that these types of problems pose. You really need to understand the problem and be creative in your solution approaches in order to get something that works. It is still a very active field of research and I think it will lead to many technological advancements and improvements in daily life down the road.

Problem Statement

Rocket League is a video game in which a player controls a rocket-powered car and tries to score a ball into a goal much like soccer. One of the things that can be done in the game is drifting. I would like to create a model that learns how to drift. The game can be played with the camera focused on the ball. Additionally, most arenas have concentric circles around the center of the field, where the ball starts, so for the purpose of this project, I think it is best to have the car drift in circles around the ball while it is in the center of the arena and use the circles on the ground to gauge the performance of the model’s drifting.

Datasets and Inputs

The main input for this problem is the screen. The screen provides information for how the car is positioned and is the primary indicator for the action that the model should take. There are screen capture libraries available that allow the screen to be loaded into python. I think it is worth reducing the screen capture to just the areas that matter, so one section would be focused on the ball while the other is focused on the car. The one focused on the car should give the model what it needs to determine the rotation of the car while the section that is focused on the ball should provide the model with relevant distance information.

The other inputs are not direct inputs to the model, but instead used to train the model to perform well on the problem. There are two different approaches to solving this problem. One is through reinforcement learning and the other is through supervised learning. The supervised learning approach requires the model to predict the action that would be taken for a given state. In this case, the actions that it would be trying to predict would my own. I would record my actions using python to detect which keys I am pressing while also recording the screen that is being displayed at the same time. The model would then use this as training data, hopefully also drifting well in the process.

The reinforcement learning approach requires a reward signal in addition to the direct input of the screen. The reward signal that I will be using will come from the metrics since in the end, the goal is to optimize the metric. I talk about the specifics of varying metrics in the metrics section.

Solution Statement

To solve this problem, I will take the screen as input to a model and try to use a Q-learning agent to get the model to produce actions that allow the car to remain drifting around the ball. The reward signal that the model gets will come from the metric. As long as the trial remains active, the model will continue getting rewards and when the car gets out of an acceptable range, the trial ends and it will no longer receive the reward. Using the bellman equation, we can come up with a prediction for future reward and the model will then attempt to select the action that receives the most reward and is most likely to make it last a long time. The model will likely be using a convolutional neural network, although there are many different ways the data can be manipulated to potentially make the agent perform better.

Benchmark Model

The simplest benchmark model is one that takes random actions. This is standard for reinforcement learning problems since we know that if we don't do better than random, we must be doing pretty bad. In reinforcement learning, random movement is normally the first step in

creating a Q-learning agent. The model is improved from there, so it makes sense for a model that selects actions at random would be good to test against.

Another benchmark model that I could use is myself. I am able to drift around the ball, although I do sometimes make mistakes, so if we can tell that the model is better than I am, we know that the model is performing very well. We could additionally use this data to create an agent that uses supervised learning to predict which action I would take and use it's score as the "computer baseline" model. All of these benchmark models would be measured in the same way as the final model, making comparison between them simple.

Evaluation Metrics

If you were trying to tell how well someone is drifting, the first metric that comes to mind is timing to see how long they last before the car veers off. I think this would also be a valuable metric in calculating the performance of a machine learning model. The longer the model is able to continue drifting the car at the proper distance, the better we can say it has performed.

Another possible evaluation metric would be to use another model to try and determine when the trial should end. It would simply classify each frame as the last frame of a sequence or not by using the data from the previous metric. This would be beneficial because it could automate the process of determining rewards for the model and allow it to train without human intervention.

As opposed to using the end of a trial for the reward signal, I could use a spectrum of rewards. +1 for being on track, 0 for being close to on track and -1 for being way off. The final evaluation for the trial would be the sum of the rewards. Additionally, I could make a regressor for this which would assign a reward to each frame and again provide the reward signal that the model needs to improve its performance automatically.

Use computer vision to determine the size of the ball and use the size of the ball to determine the distance of the car from the ball. There would be an ideal distance and the deviation from that distance would determine the reward signal, which could then be summed up to determine the score of the model.

I have one more idea for a more complex metric. The idea involves using an image similarity metric to determine how close 2 states are. Using this, we can say that the ideal state is the one closest to the average of all the states. There are many similarity metrics out there, but the simplest is to calculate the mean of each pixel and then use the squared error to determine how similar a given frame is from that mean. The mean squared error of all the pixels can be used as a negative reward signal with the end of a run having a final, very low reward signal to prevent the algorithm from just trying to end the game sooner and avoid the negative rewards. Similarly we can use PCA, auto-encoders and other similarity metrics to determine how similar 2 frames are. Of the 2 images the model gets as input, I think the ball will be more closely related to the distance, so I can also scale the errors such that deviation from the mean

of the ball image is worse than deviating from the error of the car image. Again the score of the algorithm would just be the sum of the reward signals.

Project Design

I will start by implementing the data communication pipeline between the agent and the game. After doing this, I can do data analysis and experiment with the metrics and different settings within rocket league. Then I will begin implementing the core features of the agent, making sure that the implementation is modular, so I can implement more complex features along with different internal models and reward signals. Along the way, I will need to be referencing much of the resources available online and the current research in the field. It may turn out that I need to practice implementing the Q-Learning algorithms on simpler problems before I am able to implement them in such a way that works for this task. Once I've implemented the core of the algorithm, I can start using different neural network architectures within the model and see what works. I can play around with some of the different parameters, including the exploration rate and possibly explore using different metrics to generate the rewards. I can experiment with different techniques until finally, I'll either arrive at a satisfying model or conclude that the problem is too difficult for many of the techniques we currently have available.

The data communication pipeline will consist of a few parts. The main component will be a loop that captures a segment of the screen and the keys that are pressed down. The images from the screen are preprocessed and then fed to the agent. If I decide to create a supervised learning model, the recording of the keys allows me to do that. After the agent returns an action, regardless of the algorithm, the pipeline converts it into a key press, which the game will then detect and use to update the subsequent frame.

Now that I have a means to record the screen and actions, I can start to collect and look at some of the data. Rocket League has several different arenas and many ways to customize the car, so I think that it is worth experimenting with the look of the car and the arena to make sure that there is sufficient contrast between the car, the ball and the background. I can normalize the images, convert them to black and white as well as other things. My favorite metric is using the mean squared error because it is largely quantifiable and provides a range of values, so I want to implement that first and see how well my assumptions match reality. Additionally I can find the distribution of my actions and other interesting facts.

Now I'm ready to implement the agents. First, I'll need to implement the baselines. The baselines will be a random agent and my own play. Implementing the other algorithm is going to be by far the hardest part. It is a long researched problem and currently a hot topic, so I can lean on information available online while I am designing the algorithm. It will likely be worth it to practice the techniques that I find on simpler problems to confirm my understanding before I try and implement them in the more complex setting of a videogame. One of the main considerations that I'm going to need to make while implementing the agent is it's flexibility. I will be probably be implementing many different techniques before I get anything to work, so it is

important that the interface I build is flexible. The first thing that I will be trying is Q-Learning, but it normally requires many additions before it performs well. I also need to consider that I might also be using fundamentally different action selection methods, like policy gradient and supervised learning, so the interface should be able to support all of them without too much extra hassle.

For each of the techniques, there are also many individual neural network architectures that I could use. It is probably best to define several different models and try each of them out. If the performance seems to generally be improving, I can leave it running and see how far it gets before the performance stops improving. Once the performance improvements stop, I can record the final score for that configuration and move on to the next. At this point, I have all of the resources that I need to find a model that works. If I run out of models, I can change the exploration rate and other hyperparameters, try out different metrics and make the base algorithm more complicated. Eventually I will find a model that works and have solved the problem or at least have generated enough data to highlight all of the different models that do not work.