OpenAI Autonomous Car Driving

1 Introduction

OpenAI Gym is an open-source simulation-based toolkit that allows for developing and benchmarking of reinforcement learning algorithms. Gym offers an array of environments in which an agent can explore and attempt to achieve a predefined objective. We use Gym's CarRacing-v0 environment to autonomously navigate a vehicle along a race track using a deep Q-network. The model was able to achieve an average score of about 832 out of a possible 1000 points over 100 trials.

2 Motivation

This problem does not have a clear business problem or stakeholders, as we are just developing and comparing algorithms to maximize a score. However, reinforcement learning is becoming increasingly prevalent in fields like autonomous driving, ride-sharing, and gaming. We can imagine this car-racing simulation as a mobile video game created by a gaming company that would like to develop non-playable characters (NPCs) to competitively race human-controlled players. In this contrived example, race tracks are randomly generated at run-time and have been too convoluted to develop a generalized, hard-coded algorithm to reliably usher NPCs along the track. In the interest of saving developer time and resources, the company would like to invest in a computational agent that can independently guide a vehicle along an arbitrary race track.

3 Problem and Environment

We experiment with two different environments based on the OpenAI Gym CarRacing environment. By using this ecosystem, we frame the problem as an agent-environment feedback loop that we see in many classic reinforcement learning problems. The agent will observe the environment and an associated reward signal, perform an action based on the given information, and observe the subsequent environment state and reward, and so on and so forth.

There are two different environments with which we experimented: The original environment (CarRacing-v0) and a custom, open-source environment

that added a more opinionated pre-processing framework atop of v0 (CarRacing-v1). Due to time and resource constraints, we chose to experiment only with v0 after a comparison of initial performances.

3.1 State

Each state is represented by a 96×96 grid of RGB values ($96 \times 96 \times 3$). Each element is a value between 0 and 255.

3.2 Action

The actions in CarRacing-v0 are represented by three continuous values: Steering, gas, and brake. Steering is a number in $[-1, 1]$ and gas and brake in $[0, 1]$. Because of the intractability in deciding between an infinite number of actions, we must discretize this continuous space in some simpler, meaningful way.

Through experimentation, we discretize the action space into 12 discrete actions: Do nothing, left, right, brake, brake left, brake right, accelerate, accelerate left, accelerate right, drift, drift left, drift right.

3.3 Reward

The reward function for this environment is $1000 - \frac{1}{10}N$ where N is the number of frames it takes to complete the track.

3.4 Evaluation Criteria

An environment is "solved" if it receives over 900 reward out of a possible 1000.

4 Approach

To solve this problem, we train a model to inform our agent's decisions. At a high level, this approach is as follows:

1. Reset the Gym environment and receive the current state of this environment
2. Pre-process the state into a data structure that can be ingested by our model
3. The agent takes an action, predicted by its current model
4. Store the current state, the action taken, the reward given for taking this action in the current state, the resulting state, and whether or not the episode has been terminated. Repeat and accumulate a buffer of these experiences, which we will call an experience replay buffer

5. Maintain a convolutional neural network to predict the value of taking each action from a given state. Sample a batch from this buffer to train this network. Maintain a duplicate target network to predict the value of the actions taken from the next state and thereafter.

6. Use the difference between the target predictions and the current state predictions to compute loss and adjust the weights of the first network based on this loss

7. Predict each action at each step based on the maximal value given by the network

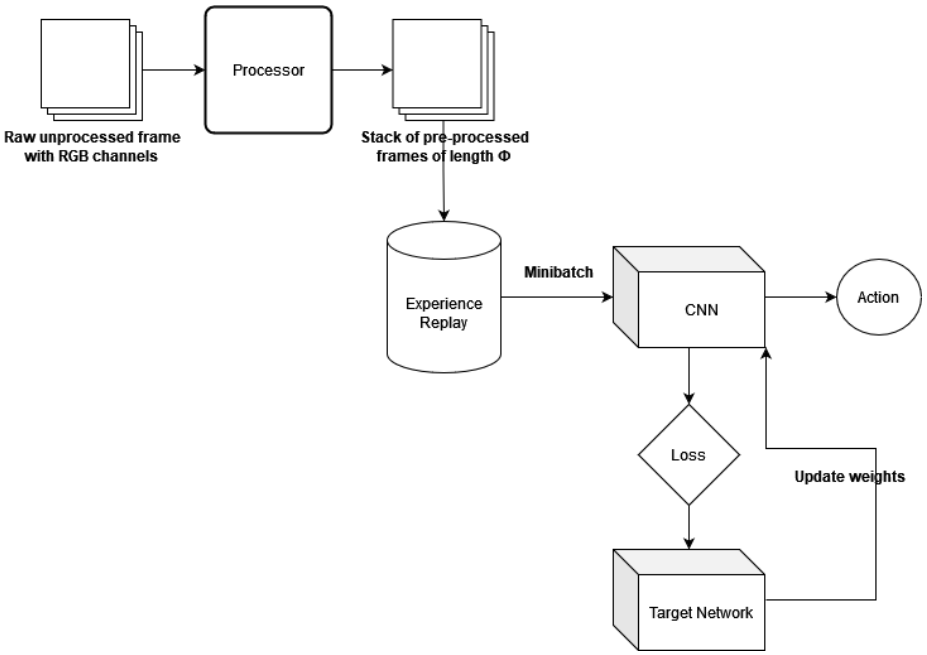

Figure 1: A system overview. Raw input enters a pre-processor, which mutates the input into a format that is more conducive for model training. The agent takes an action based on its model's prediction. It stores these experiences in a buffer, which is used to train a convolutional neural network.

We will go over these steps in greater detail in the following sections.

4.1 Pre-processing

In our preprocessing stage, we gray-scaled the frame, reducing the RGB channels to one dimension, normalized its pixel values from 0 to 1, and created a queue of consecutive frames of size ϕ. ϕ is configurable on input and attempts to capture the motion of the vehicle. Often in games with high FPS, consecutive frames

appear virtually identical. To address this, we dropped frames in between each sub-frame within the queue. Said differently, each sub-frame in the queue is exactly n timesteps away from each other. As a result, we ensure that some progress or motion is being made and properly captured by the queue.

4.2 Modeling

Modeling is accomplished through maintenance of a deep Q-network (DQN). This DQN comrpises of a convolutional neural network (CNN) that takes a pre-processed state (queue of sub-frames) as input and estimates the long-term reward of each action if taken at that state. The action with the maximal value predicted by the CNN will be the action taken by the agent on the next timestep. We follow a decaying ϵ-greedy policy, where the agent has a probability of ϵ if taking a random action, which decays over time i.e. the agent becomes more certain as time passes.

4.3 Training

Training is organized by epochs and episodes. On start, we can define the number of epochs we'd like to train the agent for and how many timesteps each epoch should have. The agent then simulates several episodes of attempting to navigate the race track. Each episode contains timesteps and eventually terminates once the vehicle completes the track. Once the sum of each episode's timesteps surpasses the defined number of timesteps per epoch, the epoch terminates and a new epoch begins.

At each timestep, the agent makes a forward and backward pass through its CNN. The loss is calculated by taking the mean squared error of the predicted action values from the CNN and the predicted action values from a duplicate CNN called the target network. The target network is identical to the main CNN but it takes the next state as input and tries to predict its best action-value. We use a second network to add stability to the predictions of the next state values. The weights of the target network are updated periodically throughout the training process.

5 Results

Over 100 consecutive trials, the agent scored an average of about 832 out of a possible 1000 reward points. Our agent does not quite reach the 900 mark but still yields a solid performance and reliably navigates the race track.

6 Recommendations

1. The training time and computational requirements needed to train an agent to navigate an arbitrary track is exorbitant and less practical than

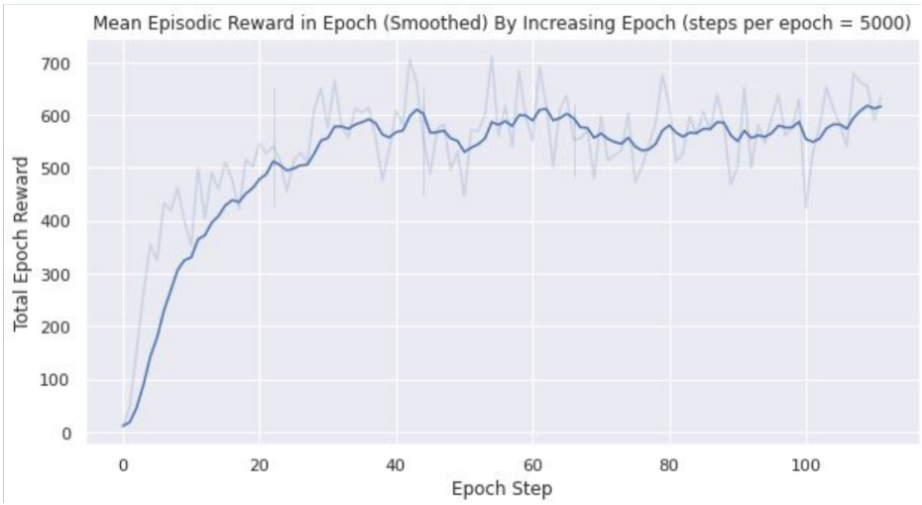

Figure 2: The mean average per episode as we progress further into training.

Figure 3: The training loss per epoch. The oscillations and non-linearities are due to stopping and restarting the agent due to memory constraints

developing waypoints or guidelines for a "less intelligent" agent in a hard-coded manner. It may be useful to explore simpler methodologies like those mentioned above in navigating simple race tracks.

2. Since the agent's only way to distinguish right from wrong is the reward signal, the agent will not necessarily emulate human-level behavior and will resort unorthodox tactics to maximize reward. The agent exhibited sharp, unhuman-like turns using its best-performing model. If this is in issue for realistic gameplay, then either (1) the reward function should be adjusted to disincentivize these types of behaviors or (2) supplant this reinforcement learning system in favor of a set of guidelines or heuristics to guide human behavior.