Deep Reinforcement Learning for Simulated Car Racing

CISC642 Final Project

Jinay Jain

Introduction

Reinforcement learning (RL) is an approach to artificial intelligence (AI) that aims to maximize a reward function by taking actions in its environment. Through several demonstrations of its success in mastering Atari games [1], playing chess [2] and Go [3], and even solving Rubik's cubes [4], it has become a widely used technique for AI research.

In tandem with the rise of reinforcement learning, deep learning has undergone a drastic increase in popularity over the past decade. Neural networks are powerful tools in approximating complex functions for tasks like classification, regression, and segmentation. Convolutional neural networks (CNNs), in particular, have become the dominant method for solving vision-based problems due to their ability to learn complex features without explicit human supervision.

Video games and simulations are commonly used as environments for training reinforcement learning algorithms, and the results of simulation have been shown to transfer well to real-world problems [5]. The goal of this project is to develop a deep learning model that can solve the problem of driving a car in a simulated environment. Using OpenAI Gym [6], a collection of simulated environments, I trained a neural network to drive a car in a simulated environment called `CarRacing-v0`. After exploring several different RL methods and simplifying the problem, I developed a vision-based model that can successfully complete the track at a high speed.[1][2]

Background

Reinforcement Learning Theory

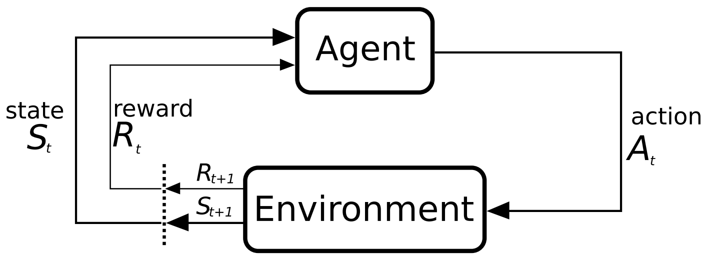

Figure 1: Visualization of RL Pipeline

[1]A full video demonstration of the algorithm is available on YouTube
[2]Code is available on GitHub

In reinforcement learning, there are two main components, the agent and the environment. The agent is some algorithm that takes in an observation of the environment s_t and produces an action a_t to take in the environment (fig. 1). The environment then changes according to that action and produces a new observation s_{t+1} and reward r_{t+1} for the action taken. The agent often has a policy π that maps the observation s_t to an action a_t. With this basic framework of reinforcement learning established, there are several different methods that can be used to train the agent to maximize the reward function.

Proximal Policy Optimization (PPO)

Proximal policy optimization (PPO) [7] is a method for training a neural network to find a function $\pi(s)$ that maximizes expected future reward. At a high level, PPO uses two neural networks, one for the policy $\pi(s)$ and one for the value function $V(s)$. As mentioned earlier, the policy $\pi(s)$ is a function that maps an observation s_t to an action a_t. The value function $V(s)$ is a function that maps an observation s_t to a scalar value which represents how advantageous it is to be in that state.

Trained in parallel, the two networks are updated as the agent interacts with the environment. The value network is trained to estimate the value of each state by minimizing the error between its predicted value and the actual value of the state. The policy network uses the estimates of the value function to select actions that are more likely to lead to higher rewards. At training time, I ran the two networks for a fixed number of iterations, and then used gradient descent to optimize the weights of the networks.

Experiments

The full code for this project can be found on the GitHub repository.

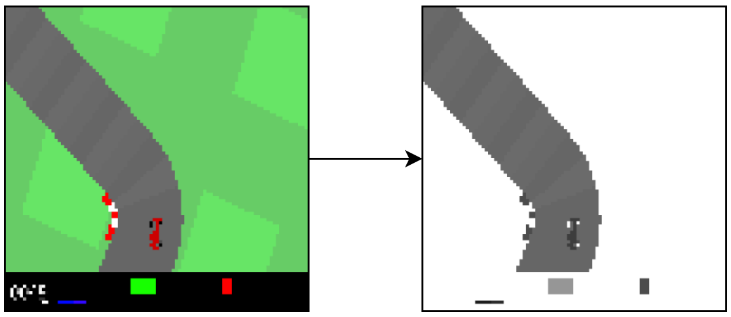

Figure 2: Frame Preprocessing Step

For the simulated car racing environment, the environment provides 96x96 RGB images of the car on the track, which includes visual indicators of the car's steering angle (green), gyroscope (red), and braking sensors (blue). As shown in fig. 2, the images are preprocessed to remove the background, hide the score, and convert the images to grayscale. This preprocessing was easy to implement using NumPy and OpenCV.

The architecture I used for both neural networks was a convolutional neural network with 3 convolutional layers and 3 fully connected layers. All activations within the network are ReLU. I implemented the models using PyTorch's `nn.Conv2D` and `nn.Linear` layers [8]. I chose the Adam optimizer [9] with a standard learning rate of 0.001 for the optimizer.

Each step of training begins with running the current version of the models for $T \geq 512$ timesteps. Using the collected data from the environment, I used the mean-squared error loss to optimize the difference between the value network's predicted value and the actual value of each state it saw. The policy network is then trained to increase probabilities of actions that lead to higher rewards, and reduce probabilities of actions that lead to lower rewards. To avoid drastic changes in the policy network's weights, I used the clipped loss function (from the PPO paper) that limits the amount of probability change that can occur in a single step. Training was done in mini-batches of size $M = 128$ for $E = 10$ epochs.

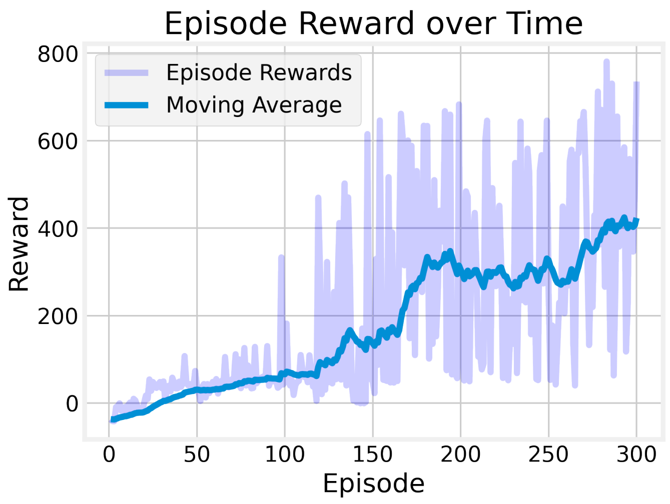

Figure 3: Training Summary

We can see that the the total reward acquired per episode over the course of training is increasing (fig. 3). The moving average of the reward is shown to demonstrate the progress of learning. In traditional machine learning, a fluctuating signal often indicates that the model is not learning properly, but in reinforcement learning, the unstable dataset can influence average reward and cause fluctuations in the training. Overall, the model is able to sufficiently perceive the track and steer the car in a reasonable way. Extra behaviors like drifting, braking before turns, and hugging the edge of the road were also observed during training.

Conclusions

This project is a demonstration of how reinforcement learning can be used in conjunction with neural networks to solve a simulated car racing problem. However, the model's sensitivity to hyperparameters and randomness was a challenge that I had to address while building the model. Future research into reinforcement learning could include finding more stable methods of learning an optimal policy, like initializing the model with a pre-trained policy from human data. This would allow the model to gain a deeper understanding of the worlds it's in without having to rely on random exploration on its own.

In this particular environment, I could help the model by using traditional computer vision techniques to detect lines and curves in the track. These higher level features could allow the model to more quickly learn how to steer the car. In fact, previous work on this environment has shown that an approach that first learns a "world model" can quickly learn how to steer in the world afterwards [10]. All in all, though, proximal policy optimization is an efficient method for training a neural network to solve this problem, and the use of convolutional neural networks allows us to generate a policy that directly maps the game images to actions.

References

[1] V. Mnih *et al.*, “Playing atari with deep reinforcement learning,” *arXiv:1312.5602 [cs]*, Dec. 2013, Accessed: Dec. 19, 2021. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] D. Silver *et al.*, “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” *arXiv:1712.01815 [cs]*, Dec. 2017, Accessed: Dec. 19, 2021. [Online]. Available: http://arxiv.org/abs/1712.01815

[3] D. Silver *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[4] OpenAI *et al.*, “Solving rubik’s cube with a robot hand,” *arXiv:1910.07113 [cs, stat]*, Oct. 2019, Accessed: Dec. 19, 2021. [Online]. Available: http://arxiv.org/abs/1910.07113

[5] M. Kaspar, J. D. M. Osorio, and J. Bock, “Sim2Real transfer for reinforcement learning without dynamics randomization,” *arXiv:2002.11635 [cs]*, Feb. 2020, Accessed: Dec. 19, 2021. [Online]. Available: http://arxiv.org/abs/2002.11635

[6] G. Brockman *et al.*, “OpenAI gym.” 2016.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv:1707.06347 [cs]*, Aug. 2017, Accessed: Dec. 19, 2021. [Online]. Available: http://arxiv.org/abs/1707.06347

[8] A. Paszke *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in neural information processing systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[9] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017, Accessed: Dec. 20, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980

[10] D. Ha and J. Schmidhuber, “World models,” *arXiv:1803.10122 [cs, stat]*, Mar. 2018, doi: 10.5281/zenodo.1207631.