Deep Reinforcement Learning in Video Games

Vigneshwer Vaidyanathan (vpvaidya@asu.edu), Bahar Shahrokhian (bshahrokh@asu.edu)
Arizona State University

Abstract—Our initial approach is to re-implement the Deep Q-Network and have it learn the policy for a simulated vehicle control game. Our choice of game is Out Run. However, many attempts at this specific category of games remove obstacles and time limits to expedite the agent's learning process. We wish to experiment with including the obstacles in the game and framing an appropriate reward function that handles this additional layer to the original problem statement.

1 Introduction

Currently, autonomous cars are the talk of the town. Various organizations, such as Google, Uber, Comma.ai, Ford, are dedicating vast resources in training and developing models for these cars to operate safely in the real world. With an eagerness to contribute to the next trend in employ artificial agents in the real world, we propose a project that deals with driving a car in a more primitive environment.

Reinforcement learning is a paradigm of learning agents that learn how to interact with their environment by trial and error**.** The agent then executes a set of trials in its environment using its policy. A policy can be thought of as a simple lookup table that the agent consults, given the current state of its environment. Every interaction (or action) that the agent performs in this environment results in a feedback. This feedback can be negative, positive, or zero and is important for the agent to learn its environment. Without a feedback about what is good and what is bad, the agent will have no grounds for deciding which move to make. As you can see, this is unlike supervised learning where the agent is taught which action to take given the state of the environment although it does receive some sort of feedback from the environment as a consequence of its actions. The task of a reinforcement learning agent is to use these observed rewards to learn an optimal (or nearly optimal) policy for the environment. Reinforcement learning can also be described as a model-free learning in that the agent has no model of the world and must "blindly" learn.

In many complex domains, reinforcement learning is the only feasible way to train a program to perform at high levels. For example, in the domain of video games, there aren't datasets that can be utilized for supervised learning. In such situations, reinforcement learning would be an optimal learning strategy.

2 Related work

Q-Learning, a learning method used to develop optimal policies, was first introduced by Chris Watkins in his Ph.D. thesis.[1] A variety of tasks in our world can be devised as Markovian Decision Process problems. The essence of these class of problems is the assumption that the current state of the agent is conditionally independent of its state history, given the its preceding state.

In 2015, a group at DeepMind successfully used a convolutional neural network to learn a Q function which successfully plays various Atari 2600 games at or significantly above professional human player ability. [2] The only inputs to their network were the images of the game state and the reward function values. The most impressive result from their algorithm was the generalizability of the approach - the algorithm learned to play a wide variety of games utilizing the same network architecture, parameters, and inputs.

Recently, although not on a peer-reviewed journal, students at the Stanford University have managed to implement DeepMind's DQN on a vehicle control game (JavaScript Racer). Their network appears to learn a stable policy for the given environment and rarely goes off-road. [3]

3 Our approach

Our initial approach is to re-implement the Deep Q-Network and have it learn the policy for a simulated vehicle control game. Our choice of game is Out Run, where the user is to keep the car on the road and the faster he/she reaches the finish line, the more points he/she gets. However, many attempts at this specific category of games remove obstacles and time limits to expedite the agent's learning process. [3] We wish to experiment with including the obstacles in the game and framing an appropriate reward function that handles this additional layer to the original problem statement. If time permits, we plan to experiment with similar games and examine the behaviour of the network. For the initial testing, we plan to utilize the framework, Universe, developed for the training of reinforcement learning agent by OpenAI. [4]

3.1 Timeline

The initial couple of weeks, we plan to implement the Deep Q Network in our preferred choice of a neural network library and run sanity checks to ensure that the agent is beginning to learn a policy. At the time of writing this proposal, both the authors have experience with constructing networks in TensorFlow but we wish to explore other frameworks, such as Theano or Torch, before we finalize on the framework we decide to use for development. [5] To start off with, we plan to have the agent learn its policy without any obstacles present in its course to achieve a baseline performance for when we decide to employ obstacles as part of the race course.

Once the model begins to learn, we hypothesize that training might take a significant portion of our project timeline. However, we do plan to experiment with variations (such as including obstacles, time limits, higher rewards for continuous acceleration, varying the discount factor as well as the reward function, etc.) to the learning during this phase if it's possible. We may require to use of the University's resources if we can't run multiple simultaneous experiments.

During the final phase of the project, we hope to visualize the results using similar algorithms as stated in DeepMind's publication and the performance of the network against various scoreboards that are publicly available. [2] We are also eager to test the model against human players to determine the performance of the model at this phase. At this time, we will finish the last of our modifications or experimentation before finalizing on the performance of our network during the various stages of the project.

4 CONTRIBUTION

As both authors are programmers and machine learning enthusiasts, we plan to divide the work in half. Since several components exist in DeepMind's DQ-network, we plan to build these components independently before integrating them and executing the network.

When it's time to experiment with the model, Vigneshwer plans to experiment with introducing obstacles in the race course and framing the reward function while Bahar will be working with integrating time limits and its needful modifications to the network's learning. However, we are not sure how effective these results are going to work out to be.

5 CONCLUSION

5.1 Utilization of material concerned with this project

Since Vigneshwer Vaidyanathan is taking a course (CSE 591: Advances in Robot Learning) which specializes in agents learning from inputs from the environment, he will be interested in utilizing this project for that course.

REFERENCES

[1] C. Watkins. Learning from Delayed Rewards, 1989. Ph.D. Thesis.
[2] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461v3 [cs.LG].
[3] J. Gordon. JavaScript Racer. JavaScript racing game we modified
[4] Universe, by OpenAI: https://universe.openai.com/
[5] TensorFlow, by Google: http://tensorflow.org/