Reinforcement Learning in MountainCart and CartPole

Mohamed Al-Thibeh, Rishhi Balakrishnan, Virack D. Chomm,
Andrew Chua, Muzammil Elahi, Shuaib Reeyaz

Wilfrid Laurier University, December 2019

An Important Problem

A fundamental goal for humanity has always been the advancement of intelligent systems. Our technology today is a testament towards this. We are approaching a critical phase in our strive for innovation; that is the development of technology more intelligent than us.

Our goal is to observe and understand the process of Reinforcement Learning. This technology aims to leverage the concept of reward to model a simple environment that eliminates the need for human understanding.

We would primarily like to see how Reinforcement Learning can be utilized to improve the performance of CartPole and MountainCar. Building an understanding of what factors lead to the successful applications of reinforcement learning.

About The Model

DQN stands for Deep Q-Learning Network. Combining a Deep Neural Network with the concept of Q-Learning. It's specifically in Q-learning where the model learns how good an action is at a particular state denoted by Q(s,a).
Often referred to as an action-value function. The model saves a memory map of all possible Q-values. That way, future iterations find it much easier to take actions to maximize the reward. Q-learning is essentially a cheat sheet of winning moves in any given situation.

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

In reinforcement learning, both the input and the target change constantly during the process and make training unstable. However, both the input and output can converge so if we slow down changes in the input and output we can model this while still allowing it to evolve.

DQN overcomes unstable learning with Experience Replay, Target Network, Clipping Rewards and Skipping Frames.

Deep Neural Networks (DNN) are easily overfitted. What ER does is store state transitions, rewards, and actions (used in Q learning) to make small groups to update the DQN. This reduces correlation between experiences by updating the DQN, leading to an increase in learning speed with small groups of data and uses past transactions to not "forget".

Implementation

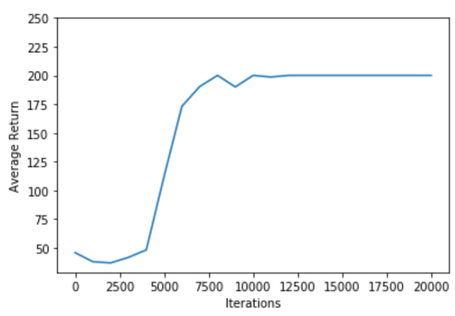
We currently have two implementations of Reinforcement learning written in Python with the use of OpenAI's Gym. The Step function in Gym returns four values:

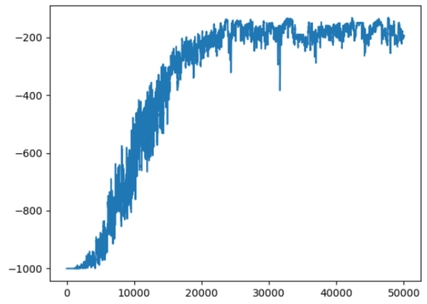
Observation represents what is happening in the environment.
Reward is the amount of reward achieved by the action.
Done is a boolean which indicates that the episode has ended.
Info is used for debugging, this is not used for learning.

A reward is given every frame the stick is upright, and the agent "dies" when the stick's angle falls below 80 degrees. This Model was written in Google's Tensorflow 2.0 using TensorFlow agents.

The neural network is a QNetwork which is a Convolutional Neural Network. The Agent is passed the environments Observations, the possible actions, the neural network, the optimizing function, the error function and the amount of steps taken.

Results





As seen in the image above on the left. In our simple environment, the model is able to consistently hit the maximum reward in 10,000 iterations. Past that, the model is capable of achieving the highest score possible without fail.

However, a game with a more complex environment, such as MountainCar, fails to achieve similar results. MountainCar situates a car between two hills, and the car has to reach the top of the right hill when the car's engine is too weak to accelerate up the right hill fully. Thus, it must instead scale the left hill and leverage gravitational potential energy to make the trip. This problem is more difficult due to a scenario with 2 variables; both position and velocity. At every instance, it must decide to move left, right, or do nothing at all, allowing gravity to take over.

As we see in the image above on the right, MountainCar takes approximately 50,000 iterations to be able to reach the high score. After that point, the model is incapable of achieving consistent results like CartPole.

Conclusion

Our goal with this project was aimed towards gaining a better understanding of reinforcement learning and the environment that this model best thrives in.

Variables such as states, actions, and rewards helped us better understand its goal and methodology. Meanwhile, we learned about the modified CNN: a DQN, and its use within the algorithm. We also looked at mainstream instances of reinforcement learning and its implementations in practical games and applications. Our team created a reinforcement learning algorithm and developed an implementation to operate within CartPole and MountainCar.

Our results show the rate at which our algorithm masters these games. CartPole took about 10,000 iterations to achieve the high score consistently. MountainCar on the other hand could not achieve consistency, even with 50,000 iterations. The added variables and complexity of MountainCar made our model less accurate, though it still consistently made improvements upon iterations.

Acknowledgements

Reinforcement Learning is still a relatively new field so the applications are potentially limitless. Reinforcement Learning is computationally cheap and can be extended to sports like tennis or to self-driving cars.

The continuous learning model that reinforcement learning is based on is excellent for environment's that remain consistent. However, this particular model does not adjust well to the introduction of random or unexpected situations.

References

1.DeepLearning.TV,director.Reinforcement-Learning-Ep.30(Deep-Learning-SIMPLIFIED).YouTube,YouTube,15 Sept. 2016, www.youtube.com/watch?v=e3Jy2vShroE.
2.Silver, David, and Demis Hassabis."AlphaGo Zero: Starting from Scratch." Deepmind, Deepmind, 18 Oct. 2017, deepmind.com/blog/article/alphago-zero-starting-scratch.