Applying Q-learning to Continuous

State-Action Spaces Using Discretization

David Tandetnik

Northeastern University
tandetnik.da@northeastern.edu

Abstract

Q-learning is a common technique in reinforcement learning that is used to train agents in a discrete state-action space. In this paper, it is applied to a continuous state-action space with the use of discretization. Namely, the algorithm is run on a simulated robot hand environment provided by OpenAI Gym and the results are compared to baselines achieved with more complex algorithms like DDPG published by OpenAI researchers Matthias et al., 2018.

Introduction

Training agents using Q-learning is a common and effective technique in reinforcement learning. The algorithm has relatively small computational requirements and often performs well for basic agents. Q-learning is however restricted by its need to hold a two dimensional table in memory with a cell for every possible state-action pair in the RL environment. For environments with continuous state-action spaces, this proves to be an intractable approach to training agents. Algorithms such as Deep Deterministic Policy Gradient (DDPG) exist for solving these environments, and indeed they essentially employ the core idea of Q-learning but substitute the need for a large state-action table with neural networks that approximate the values normally provided by the table (Lillicrap et al., 2015). These approaches however tend to be much more computationally expensive and therefore time-consuming to train. This paper instead explores whether it is possible to achieve similar levels of success on a continuous state-action space with Q-learning as it is with DDPG. The solution is rooted in the process of discretization, where the continuous state-action space is separated into discrete intervals/values, which in turn makes basic Q-learning at least theoretically feasible.

To compare the different training methods, an OpenAI Gym environment that simulates a robot hand was used (Matthias et al., 2018). The environment, "HandManipulateBlock"[1], tasks a robot hand holding a block to manipulate the block into a target position and rotation. On its own, the environment is interesting because of how it directly maps to real-world scenarios of robot automation; the ability to have a robot hand move a box to an arbitrary position would likely be useful in industries such as shipping and manufacturing.

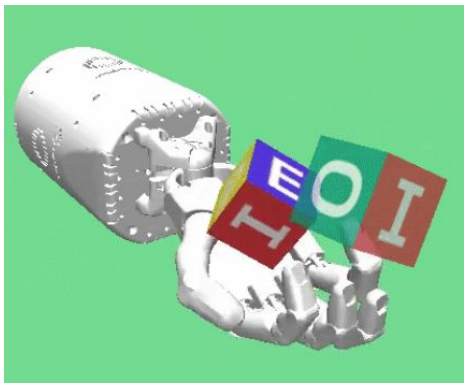

Fig. 0. Graphical render of "HandManipulateBlock" OpenAI Gym environment.

[1] https://gym.openai.com/envs/HandManipulateBlock-v0/

Background

Fundamentally, Q-learning works by using observations obtained in simulated runs of the environment ("episodes") to assign values to pairs of environment states and actions. These values represent the expected future sum of rewards if the agent takes that action in that state. The algorithm's efficiency in its training comes from the fact that for every state-action pair it computes a value for, it makes use of values computed in previous episodes and adjusts based on the reward it receives in the current episode. In order to "remember" these previous values after every episode, the algorithm requires the use of a two-dimensional table of size |state space| times |action space|. The larger the state and action spaces are, the more memory the algorithm requires and the more episodes it needs to train in order to learn meaningful actions. Naturally, if the state and action spaces are infinite (continuous), then the algorithm is virtually unusable.

More advanced techniques like DDPG overcome this issue by removing the need for this table entirely. Rather than remembering exact values from previous episodes, DDPG trains neural networks, which do fine with modelling continuous functions, alongside learning the Q-values in order to approximate rewards from previous episodes. The cost however is increased computational complexity relative to the far more simple Q-learning.

The approach in this paper will be to take an OpenAI Gym environment that by default has a continuous state-action space and discretize it into finite values. The finite values serve as "buckets" that generalize an interval of state-action pairs and allow basic Q-learning to maintain its two-dimensional table.

Related Work

The OpenAI Gym environment explored in this paper, "HandManipulateBlock", was released alongside research done by Matthias et al. in 2018. In their research, they attempted to train RL agents in a variety of different robot hand environments using DDPG and a technique called Hindsight Experience Replay, a method that improves the performance of algorithms like DDPG in environments with sparse rewards (Andrychowicz et al., 2018). Their results will be compared to what was achieved using Q-learning and discretization in this paper.

Research by Arts in 2017 explored applying discretization to Q-learning in continuous state-action spaces within the context of the videogame Minecraft. His experiment achieved reasonably high success, but the environment in his research is arguably simpler than the OpenAI Gym environment explored here. For example, his discrete action space had between 4 and 8 dimensions, whereas the approach discussed here will be attempted with 10. Additionally, he did not provide any comparisons with more advanced algorithms like DDPG, whose relative performance is the focus of this paper.

Project Description

The OpenAI Gym environment "HandManipulateBlock" simulates an environment in which a robot hand is set in an initial random position holding a perfect cube that is also set in a random orientation. The goal of the environment is to provide a series of actions (i.e. hand joint movements) that manipulate the block into a certain target position and rotation. Each simulation (episode) runs for 100 timesteps, where each timestep represents a single action being performed. After every timestep, if the block is within a certain degree of error from the target position, the agent receives a reward of 0, else the agent receives -1.

This environment is actually divided into 4 sub-problems, namely the goals of achieving proper block rotation along the Z axis, the XY axis, the XYZ axis, and then finally achieving proper rotation along the XYZ axis as well as proper position in 3D space. For this experiment, the agent is only tasked with achieving proper rotation along the Z axis.

The state space of the environment consists of a vector of 61 arbitrary floats, mostly between -1.0 and 1.0. These floats represent the robot hand's 24 joints' positions and velocities, the block's position, velocity, and rotation, and the block's target position and rotation. For the discretization process, the state space was reduced to just the block's current and target rotation along the Z axis. The OpenAI Gym simulator uses quaternions as opposed to euler angles to represent rotation, so each Z rotation is represented by two numbers. Thus the reduced state vector consists of four floats, which were also capped between -1.0 and 1.0 and rounded to the nearest tenth. The result was a reduction from a continuous state space to one with roughly 200,000 different variations.

The action space of the environment on the other hand normally consists of a vector of 20 arbitrary floats, representing the absolute position of the hand's joints the hand should move to. (The hand has 24 joints, but 4 are coupled, so only 20 are actually directly movable.) For the discretization process, the first ten of these values were set to be either -0.25 or 0.25, and the second ten were set to 0.

In order to account for the fact that the state space no longer contains information about the hand's current position, these values were changed to represent relative target positions of the hand's joints. In other words, -0.25 and 0.25 were deltas applied to the current positions, rather than absolute positions. Thus the reduced action space consists of 20 floats, 10 held at 0 and 10 taking on one of two values, reducing the continuous action space to one with 1024 different values.

The resulting state-action table had a size of roughly 10^8. All of the subsequent experiments were run using this table.

Results

The first experiment run explored different Q-learning parameters. In Q-learning, there is a learning-rate parameter alpha, and a discount factor gamma. For these tests gamma was held at 0.9 and alpha was tested at 0.05, 0.1, and 0.2. The simulation was run for 19,000 episodes, grouped into 10 epochs. 1900 episodes per epoch was a somewhat arbitrary choice but was used to align with Matthias et al.'s DDPG setup in order to allow for comparisons down the line. After each epoch, 100 test episodes were also run to evaluate the agent's performance. (Training does not occur during the 100 test episodes.) The average sum of rewards as well as test success rates are shown below for each epoch.

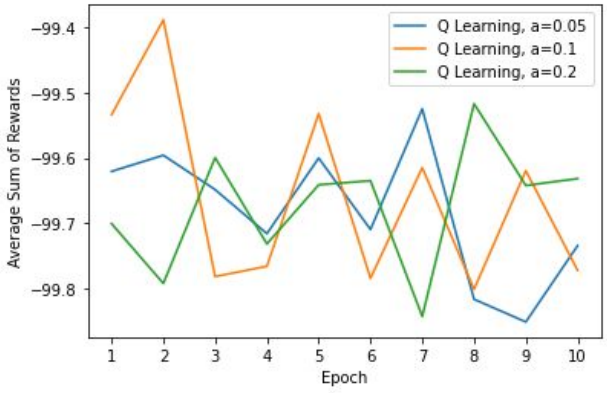

Fig. 1. Sum of rewards per episode, averaged across epochs, using Q-learning and alpha values of 0.05, 0.1, and 0.2.

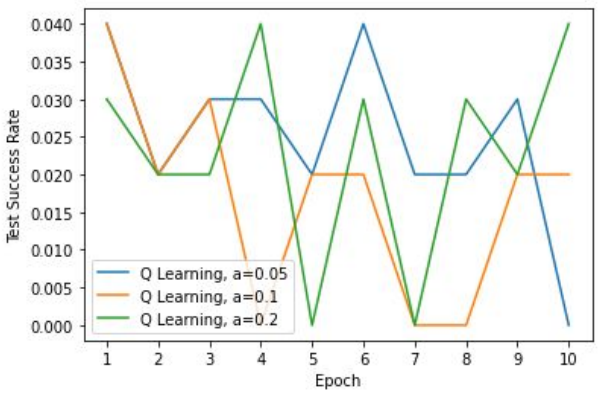

Fig. 2. Success rate of 100 test episodes run after each epoch. An episode is considered successful if it contains at least one non-negative reward.

The parameter search results do not identify a clear best alpha value. All three have similar variation, maximum, and minimum values for both the average sum of rewards and the test success rate metrics. An alpha of 0.05, shown in blue in Figures 1 and 2, seemed to have the smallest variation ultimately, and so was chosen as the best parameter for the main experiment. Note that the highest test success rate achieved for using any of the three parameters was 0.04, whereas Matthias et al. saw a rate of about 0.2 by epoch 10 using DDPG on the same environment.

The main experiment consisted of running Q-learning on the same discretized state-action space with an alpha of 0.05 for 50 epochs. The average sum of rewards per epoch and the test success rate per epoch are shown below.

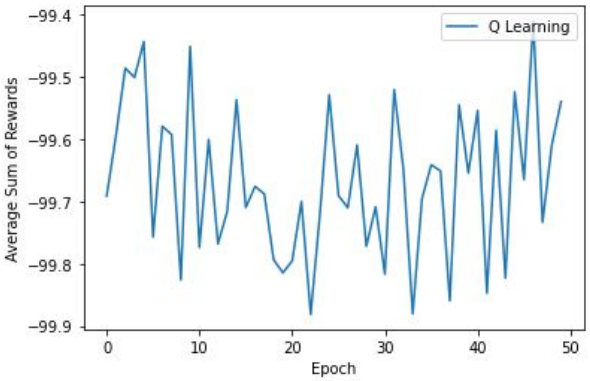

Fig. 3. Sum of rewards per episode, averaged across epochs, using Q-learning with alpha 0.05 and gamma 0.9

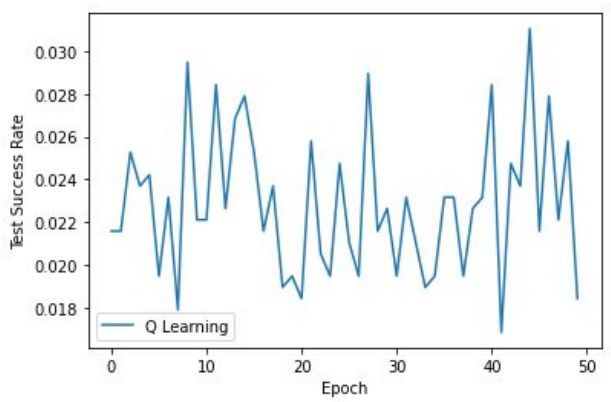

Fig. 4. Success rate of 100 test episodes run after each epoch. An episode is considered successful if it contains at least one non-negative reward.

The results show that the agent did not manage to improve its performance simply by being exposed to more training. The average sum of rewards per epoch remained between -100 and -99.4, as with the parameter search experiment. The maximum test success rate also did not increase past 0.04. For comparison, Matthias et al. achieved a test success rate of about 0.8 by epoch 25.

Conclusions

Ultimately the experiments performed in this research show that applying discretization to Q-learning in continuous state-action spaces is theoretically possible, but will almost always be out performed by dedicated techniques like DDPG. The tradeoff between performance and computational complexity will vary from environment to environment. For "HandManipulateBlock" the current setup suffers from a state-action that is still too large.

The 50 epochs that were run cover less than 100,000 episodes, whereas the state space alone has almost 200,000 different values. Clearly the state and action spaces need to be reduced further, but this may come at a further cost of performance. Alternatively, the agent could be trained for significantly more episodes, and this may eventually result in better performance, but may also end up just as time-consuming and resource-intensive as DDPG.

References

Matthias, et al. "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research." ArXiv.org, 10 Mar. 2018, arxiv.org/abs/1802.09464.

Arts, Luuk. Comparing Discretization Methods for Applying Q-Learning in Continuous State-Action Space. Radboud University, 30 June 2017.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, Continuous control with deep reinforcement learning, CoRR abs/1509.02971 (2015).

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, Wojciech Zaremba, Hindsight Experience Replay, ArXiv.org, 23 Feb. 2018, arxiv.org/abs/1707.01495