Reproducing Optimistic Exploration Even With a Pessimistic Initialization

Duncan Mays
Queen's University
16djm1@queensu.ca

Reproducibility Summary

Scope of Reproducibility

The paper Optimistic Exploration Even with a Pessimistic Initialization by Rashid et al. [1] presents a mechanism for optimism and exploration in deep Q learning agents that is completely separate from the neural network contained within them. It is represented by a quantity which is added to the q values of each state/action combination as predicted by the q-network during action selection, and encourages the agent to explore the environment by making the agent optimistic towards novel state/action combinations. The authors prove its efficiency in the tabular setting, and show its effectiveness in the deep reinforcement learning setting by running trails on three different reinforcement learning challenges. This reproduction aims to replicate the results that the authors achieved in the deep reinforcement setting.

Methodology

The authors of the original paper assessed the effectiveness of their method on three trials: a synthetic Markov environment simply known as Random Chain, a maze problem, and the notorious Montezuma's Revenge. They conducted a grid search across a set of hyper-parameters to determine the optimal configuration for their method in each of the three environments and conducted extensive baseline trials to determine the effectiveness of other state of the art techniques. They then compared the effectiveness of the optimal configuration of their method to the baselines.

In this reproduction report, we will recreate the grid search on hyper-parameters, but not conduct baseline tests again. We will determine a set of optimal hyper-parameters, independently of the original paper, and then compare the effectiveness of the resulting agents to the results found in the baseline tests. This is effectively testing two hypothesis, that the hyper-parameters determined in the original publication are indeed optimal, and that their method outperforms other state of the art techniques as claimed.

Results

I do not have results to report at the moment. I have begun work on the Random Chain trial, but have not written any code for the Maze and Montezuma's Revenge Trials. The reader should be aware that good results will take weeks to produce. In the Maze trial, performing backpropegation on a single batch of transitions takes around 250 milliseconds. Updates happen after every timestep, and with the one million training timesteps that the original authors used, a single run takes 70 hours.

What was easy

Implementing the static hashing function was very easy. The authors mentioned that it was an important challenge, but they probably meant that developing a solution to the problem was difficult, not that implementing that solution would be difficult.

What was difficult

Implementing Q-learning was surprisingly difficult. It is very easy for the neural network to become unstable, minor changes to hyper-parameters can produce stability issues.

1 Introduction

Encouraging exploration is an effective way to make reinforcement learning agents more generalizable and performant. Agents can get "stuck" in a non-optimal pattern of behavior when they have not fully explored their environment, because there may be another pattern of behavior that will lead to greater reward, but that they have not discovered yet. Encouraging exploration forces agents to adopt generalizable strategies that can function in varied states, and dramatically increases the likelihood of the agent discovering the optimal strategy.

Methods to encourage exploration have not been applied effectively in non-tabular domains, where complex, high-dimensional state-spaces force the use of deep reinforcement learning techniques, such as deep Q-learning from [2]. Neural networks can be initialized to output high q estimates for unvisited state/actions, but these high evaluations usually do not last far into training because the Q-netowrk will generalize negative results between unrelated state/actions in early training. A neural network will generalize low q values in states it has visited to states it has never been to, removing the optimism of the initialization.

Moreover, most modern methods take an intrinsic approach to encouraging exploration, that is, they add a term to the agent's reward that represents the novelty of the previous state/action. This is ineffective because it rewards the agent only after it has already performed a novel state/action, not before it has, and so it relies on random noise in the system to drive the system to novel state/actions. The method we reproduce here takes an active approach, increasing the q value of novel state/actions even before the agent has been to them. The method relies on effective ways to count the number of visits to state/actions.

In complex, high-dimensional state-spaces, developing an effective way to count the number of times an action has been taken in a certain state is an important challenge. Various methods have been given in the original paper, including static hashing of the state-space, as well as simply down-sampling an image of the environment.

1.1 Optimistic Pessimistically Initialized Q-Learning

In deep Q-learning, a neural network is trained to approximate the q values of a given state, that is, the expected reward of taking each action plus the discounted future rewards made possible by that action. The agent can then select the action with the highest q value to maximize its total reward across multiple time steps. let $Q(s,a)$ be the q value of state s and action a, as estimated by a neural network. The authors augmented this function with an auxiliary term, creating a new state/action valuation function:

$$Q^*(s,a) = Q(s,a) + \frac{C}{(N(s,a)+1)^M} \tag{1}$$

Where C is a constant hyper-parameter, determining the proportional weight of the second term, N(s, a) is the number of times that the action a has been taken in the state s, and M is another hyper-parameter, determining the rate of decline of the value of the second term with the number of times the state/action has been visited. The second term in the above equation is meant to represent the novelty of the state/action (s,a). In Optimistic Pessimistically Initialized Q-Learning (OPIQ), the agent will choose the action that maximizes $Q^*(a,s)$, not $Q(a,s)$ as in standard deep Q-learning.

2 Scope of Reproducibility

This reproducability study aims to recreate the results achieved by the original authors by using OPIQ in a deep reinforcement learning setting. We will not reiterate the proofs of efficiency in the tabular setting. Due to time constraints, we will also not recreate the extensive baseline tests that the authors of the original paper conducted. The claims from the authors of the original paper that we will reproduce can be summarized as:

- The hyper-parameters that they listed are optimal for the trail environments as claimed
- OPIQ explores more effectively than each of the baselines
- OPIQ performed better than each of the baselines

3 Methodology

To conduct this study, we will write an implementation of deep Q-learning with OPIQ in Python, using the framework Pytorch. The static-hashing function, novelty reward function, and bootstrap regime will be recreated from scratch. We will also re-implement the Random Chain environment, since it is extremely simple. We will use python libraries to emulate the other environments. Maze can be emulated in OpenAI Gym [3], and Montezuma's revenge can be emulated in Arcade Learning Environment [4].

We will recreate the grid search that the authors conducted on hyper-parameters for each environment. The authors provided the set of values that they searched over for each trial. This will be done to verify that the hyper-parameters that the authors provided in the original paper are indeed optimal for each environment. Since the set of hyper-parameters under consideration is a discrete set, our results should exactly match the authors. Hyper-parameter sets will be compared based on the total, aggregated reward that the trained agent receives in a set number of episodes.

For each set of optimal hyper-parameters, we will conduct a series of training runs on each environment. For each episode, we will record the performance of the agent in terms of its reward, and some metric on the number of state/actions it has visited during training. The main metric we will use in this respect is the percent of state/actions that the agent has visited more than a set number of times, as recorded by $N(s, a)$. We will also use some trial specific measurements, such as the number of rooms visited in Montezuma's Revenge.

The data we gathered on the performance and exploration of each agent will then be compared to similar data on the baseline trials. The total reward the agent achieved per episode will be used to assess the agent's performance, and the number of states visited will be used to assess the agent's exploration rate.

4 Environments

4.1 Random Chain

Random Chain is a completely synthetic environment with a 1 dimensional state space of 100 states, arranged in a line. The agent starts at position 2, and can move either left or right along the chain. If it moves left at position 1, it receives a reward of 0.001, and is reset, if it moves right at position 100, it receives a reward of 1, and is reset. There are no other rewards given. Should the agent not reach a terminal state, the environment will reset after 109 timesteps. This environment is designed to test the agent's exploration and planning abilities, by seeing if it will move 100 times right to get a reward of 1, rather than twice to the left to get a reward of 0.001.

4.2 Maze

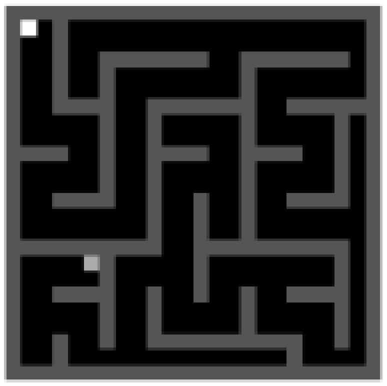

Figure 1: An image of the maze environment

In the Maze environment, the agent must navigate through a maze in a 24x24 gridworld. It starts at a randomly set location in the maze at the beginning of every episode, and then it must find a goal hidden at the same spot every episode. Should it reach the goal, it will be given a reward of 10, no other rewards are given in the game. Should it not reach the goal, the environment will be reset every 250 timesteps.

4.3 Montezuma's Revenge

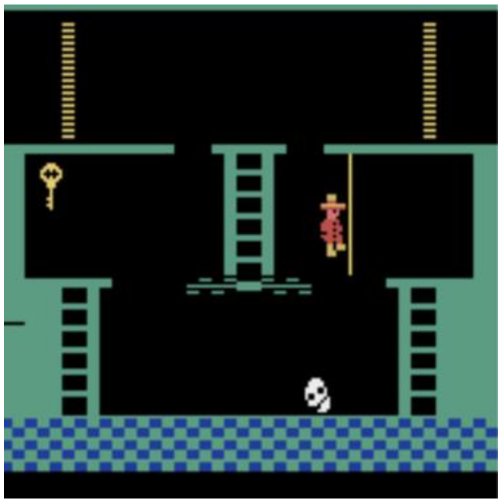

Figure 2: An image of Montezuma's Revenge

Montezuma's Revenge is a classic Atari game where the agent must navigate in between enemies, and interact with its environment in complex ways like climbing a ladder. The agent must find a key and bring it to a door. This task is notoriously difficult for current reinforcement learning algorithms, due to the complexity of navigating the environment and the prevalence of enemies. It is also difficult because, without proper curiosity, it is unlikely that an agent will pick up the key or go to the door, let alone both in the same episode and in the right order for the door to open.

All models will be trained by RMSProp with a learning rate of 0.0005, and had a discount factor of 0.99. However, the authors used a more complicated training regime on Montezuma's Revenge. The first change they made was to use Mixed Monte Carlo return, they also did not enter terminal states into the agents replay memory. They also used something called sticky actions, which means the agent had some probability p of selecting its last action, regardless of state. In the original paper, p was set to 0.25 for this trial.

5 Model descriptions

5.1 Random Chain

The deep Q-learning agent that was used on the Random Chain trial had 2 layers of 256 ReLU activated units. Its input was a 1x100 one-hot vector representing the position of the agent along the chain, and its output was a 1x2 vector containing the q scores it estimated for the actions of moving left and right.

5.2 Maze

The deep Q-learning agent that was used on the Maze trial had 2 convolutional layers, each with 16 3x3 filters that moved at a stride of 2. It then had 2 fully connected layers with 400 and 200 units respectively. All layers had ReLU as their activation function. Its input was a 24x24x1 tensor representing an image of the maze. In that tensor, 0 represents empty space, 1/3 represents a wall, 2/3 represents the goal and 1 represents the agent. Its outputs are 1x4 vectors containing the q scores it estimates for the actions of moving left, right, up and down.

5.3 Montezuma's Revenge

The deep Q-learning agent that was used on the Montezuma's Revenge trial had 3 convolutional layers, and 1 fully connected layer. The first hidden layer had 32 8x8 filters at a stride of 4, the second 64 4x4 filters with a stride of 2, and the third had 64 3x3 filters at a stride of 1. The output of this layer is then flattened and fed into a layer with 512 units that gives a 1x4 output, representing the estimated q values of moving up, down, left, and right.

The input to the neural network consists of an 84x84x4 image. Each 84x84x1 slice of this tensor represents one of 4 consecutive frames of the screen, the value at each index is the max RGB value of the corresponding pixel in the image.

References

[1] Tabish Rashid, Bei Peng, Wendelin Böhmer. 2020. Optimistic Exploration even with a Pessimistic Initialization

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver. 2015. Human-level control through deep reinforcement learning

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson. 2016. OpenAI Gym

[4] Marc G. Bellemare, Yavar Naddaf, Joel Veness. 2012. The Arcade Learning Environment: An Evaluation Platform for General Agents