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 (OPIQ) 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 bootstrapping, 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 trials on three different reinforcement learning challenges. This reproduction aims to replicate the results that the authors achieved in two of the three trials in the deep reinforcement learning 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 and ablation trials to determine the effectiveness of other state of the art techniques. They then compared the effectiveness of the optimal OPIQ configuration to each baseline and ablation.

In this reproduction report, we will recreate the grid search on OPIQ hyper-parameters and baseline tests on two of the three environments. 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 the baseline as claimed.

Results

The hyper-parameters that resulted from our grid search of the Random Chain environment did not match those found by the original authors. Moreover, we found that OPIQ only marginally outperforms the baseline method on Random Chain.

What was easy

Getting the environments working was very easy. In the case of the maze trial the original author's code could be copied and used with almost no alteration.

What was difficult

OPIQ relies on pseudo-counts of state/action combinations, which relies on hashing state/action combinations which the original authors implemented via a matrix multiplication and sign operation. Implementing this in an efficient way was surprisingly difficult. We settled on using a serial operation that needs to happen every training loop, and it slows training down considerably.

Submitted to ML Reproducibility Challenge 2020. Do not distribute.

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, 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 visited 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.

1.2 Static Hashing

Developing effective methods to count the number of visits to each state/action in such high-dimensional state-spaces is a crucial problem in implementing OPIQ. The original paper, and this reproduction of that study, used a method that the authors called static hashing. Each state/action pair is flattened and concatenated into a single, flat tensor z. This tensor is then multiplied by a randomly set matrix A, and the sign of the output vector is taken. This binary list is then treated as a hash that can map through a dict to an integer representing the number of visits to the starting state/action.

$$hash = sign(Az) \tag{2}$$

The output size of the matrix A is considered to be the size of the hash. If A outputs 32 features, then the hash is said to be 32 bits, since there are 32 independent numbers that could either be zero or one.

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 performed better than each of the baselines

These claims will be verified in only two of the three environments used by the original authors: the Random Chain trial and the Maze trial.

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 the original author's code to emulate the Maze environment.

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 at the end of training.

For each set of optimal hyper-parameters, we will conduct a series of training runs on its environment. This will be done to compare the performance of the optimal OPIQ configuration with the baseline, which is standard deep Q learning. Metrics for success will not only be the total reward each agent earns over some range of timesteps at the end of training, but also the rate at which it learns about its environment, as this is of significant practical concern.

3.1 Experiments

3.1.1 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.

The hyper-parameters that we will grid-search over are:

- $M \in \{0.1, 0.5, 2.0, 10.0\}$
- $C_{action} \in \{0.1, 1.0, 10.0, 100.0\}$
- $C_{bootstrap} \in \{0.01, 0.1, 1.0, 100.0\}$

Some other important details are:

- training lasted 100,000 timesteps
- the agents used Double-Q learning where the main model was updated every 200 timesteps
- the agents used an epsilon-greedy algorithm, with epsilon set to 0.01
- batch size was 64
- hash size was 32
- replay buffer stored 10,000 transitions
- training started when replay buffer reached 1,000 transitions

3.1.2 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.

- $M \in \{1.0\}$
- $C_{action} \in \{0.1, 1.0, 10.0, 100, 0\}$
- $C_{bootstrap} \in \{0.01, 0.1, 1.0, 10.0\}$

Some other important details are:

- bootstrapping used 3 steps, rather than 1
- training lasted 1,000,000 timesteps
- the agents used Double-Q learning where the main model was updated every 1,000 timesteps
- the agents used an epsilon-greedy algorithm, with epsilon set to 1.0 at the beginning of training, and then decreased linearly to 0.01 over 50,000 timesteps
- batch size was 64
- hash size was 128
- replay buffer stored 250,000 transitions
- training started when replay buffer reached 25,000 transitions

All agents were trained by RMSProp with a learning rate of 0.0005, and had a discount factor of 0.99.

3.2 Model descriptions

3.2.1 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.

3.2.2 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.

3.3 Computational Requirements

We trained the agents on one of our personal desktop computers, containing a Ryzen 7 3700X with 8 cores at 4.1 GHz, and a RTX 2070 super with 2,560 CUDA cores and 8 GB GDDR6. The table below gives update times and training times for environments and agents. The update time is considered to be the amount of time it takes for a single step in the environment, and then for the agent to sample a batch of transitions from its replay memory, calculate the correct q values (including OPIQ), and train its neural network on the data.

Environment	Agent Type	Update Time	Total Training Time
Random Chain	Deep Q	2 ms	8 minutes
Random Chain	OPIQ	5 ms	21 minutes
Maze	Deep Q	30 ms	8.5 hours
Maze	OPIQ	50 ms	14 hours

4 Results

4.1 Random Chain

By comparing the average total reward each agent earned in the last 50 episodes of 5 we determined that the optimal configuration of OPIQ parameters was:

$$M = 0.1, C_{action} = 0.1, C_{bootstrap} = 0.1 \tag{3}$$

This is in contradiction to the results of the original paper, which gave:

$$M = 0.5, C_{action} = 1, C_{bootstrap} = 1 \tag{4}$$

as the optimal hyper-parameters. The differences in our results do not stop there. Below is a plot of the average reward of each agent type in the Random Chain trial averaged over 9 different training runs.

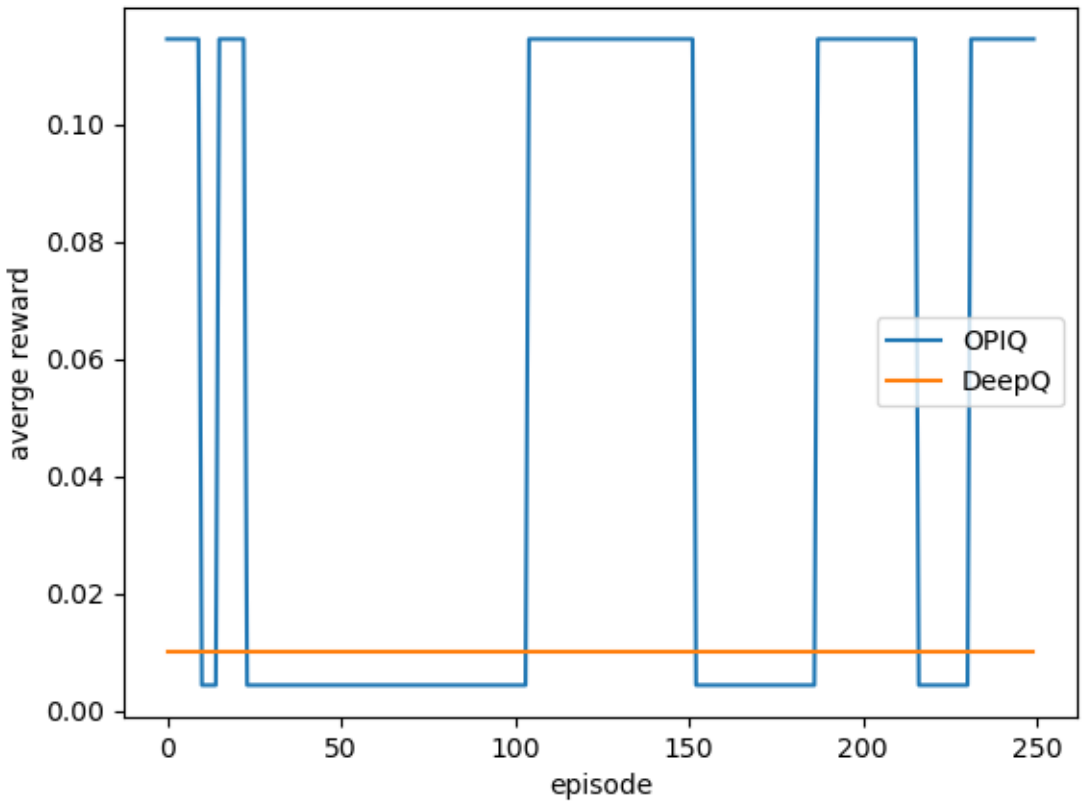


Figure 2: A plot of the rewards of OPIQ versus DeepQ on the Random Chain environment

We can see that although the OPIQ agent does, on average, score higher than the baseline, it still does not solve the problem reliably. Remember that in the Random Chain environment, the agent must choose to move 100 links down the chain for a reward of 1 rather than a reward of 0.001 that is right next to it. The DeepQ agent chooses the latter option every time, whereas the OPIQ agent seems to occasionally wander to the other end of the chain.

4.2 Maze

Sadly, due to time constraints and poor planning, we were not able to get any reliable results for the Maze trial.

5 Discussion

5.1 Unequal Results

Our results did not match those of the original authors. The hyper-parameters we found for Random Chain were different, and the resulting agent's performance was significantly lower than reported in the original paper. Our approach was very weak in the sense that we only recreated one of the the author's study, and so this may be due to some effect that is unique to the Random Chain environment.

5.2 Difficulties in Implementation

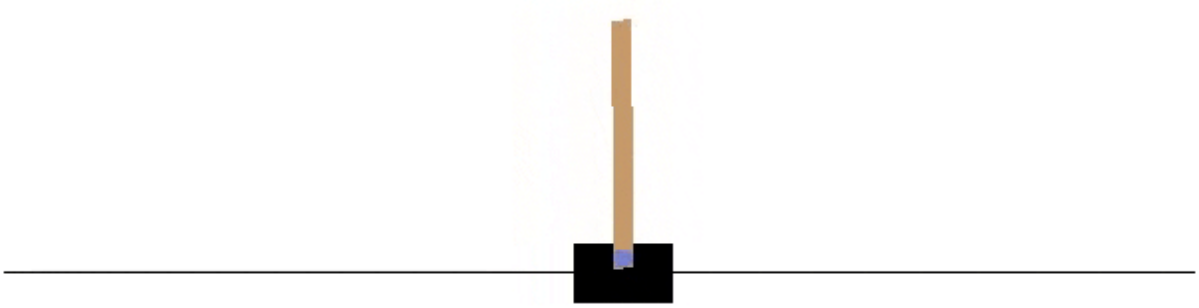
There are several practical concerns with OPIQ. Firstly, it is slow. Every training run requires the agent to look up the number of visits to each state/action, and it is very difficult to do this operation in parallel, rather than in serial. In the Random Chain trial, the agent used 32 bit hashes. If the size of an integer is 4 bytes, and there are 2^{32} integers we need

to keep track of, than this would consume 2^{34} bytes or 16 GB. The amount of memory needed to run the Maze trial in this case, which used 128 bit hashes, would be 64 TB. To prevent this memory explosion, we must use a dict to map hashes to counts, not an array. This is because dicts are only as large as the key/value pairs they contain, and the hashes are very sparse so we only end up using a small fraction of the total number of hashes. This has the implication that any algorithm that alters state/action counts in parallel would have to implement locks and other thread stability measures to prevent collision errors between threads. And so we must perform operations on state/action counts in serial, significantly increasing training time. Refer to the chart in section 3.3 for a description of update times with and without OPIQ.

Moreover, the successful implementation of OPIQ requires a grid search over hyper-parameters. Not all hyper-parameter configurations work, some do not have any effect on performance. Anyone using OPIQ in a practical setting would have to search over a space of hyper-parameters to know that the hyper-parameters they were using were working.

5.3 Cartpole Environment

OPIQ is not true curiosity in my mind, it does not rely on the inability of the agent to predict the an outcome, but only on state/action counts. I thought this might lead OPIQ to fail in some trials, so I ran OPIQ in a third environment.



An extremely common reinforcement learning environment is OpenAI Gym's CartPole-v0, where and agent must move a cart back and forth to balance a pole on it. The goal is to keep the pole upright for as long as possible. This environment is useful for our purposes because the goal is to stay in a steady state for as long as possible, and yet OPIQ is tuned to move to new states as much as it can. I ran the gridsearch over the following parameter space:

- $M \in \{0.1, 0.5, 2.0, 10.0\}$
- $C_{action} \in \{0.1, 1.0, 10.0, 100.0\}$
- $C_{bootstrap} \in \{0.01, 0.1, 1.0, 100.0\}$

and found that the optimal parameters were:

$$M = 0.5, C_{action} = 0.1, C_{bootstrap} = 0.1 \tag{5}$$

Averaging rewards over 5 training runs for both OPIQ and a baseline DeepQ agent, the total reward, which is the number of timesteps that the cart is able to balance the pole, in each episode is plotted below:

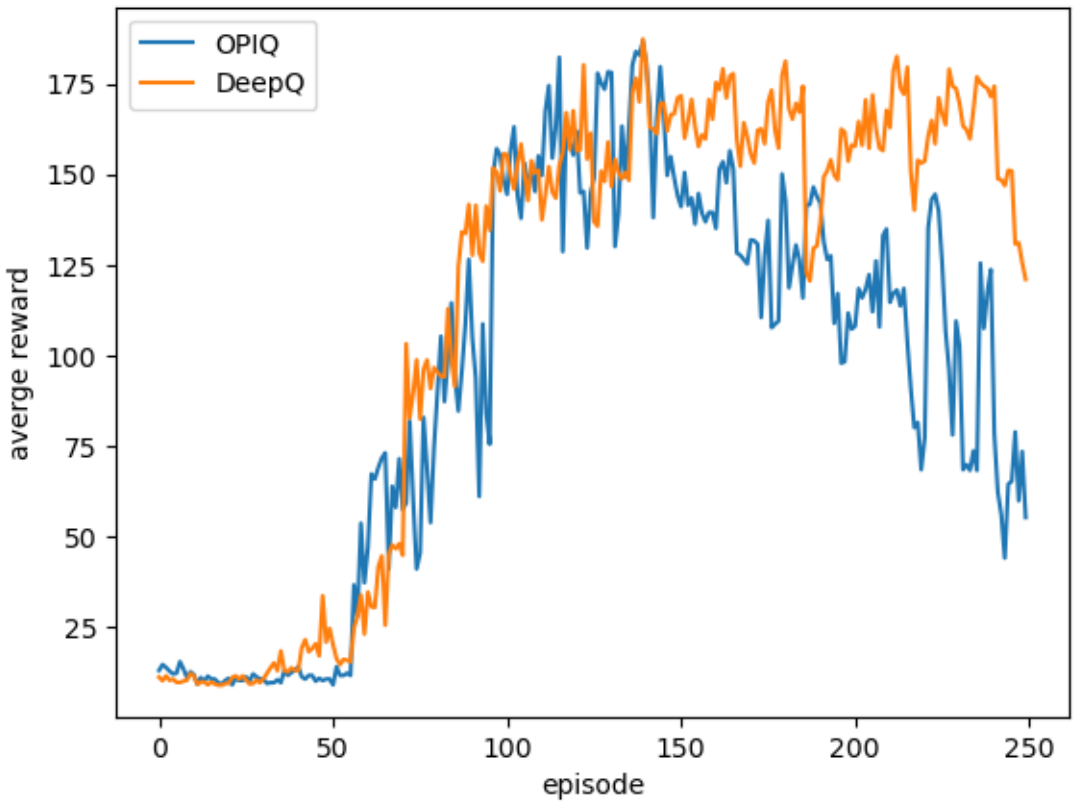


Figure 3: A plot of the rewards of OPIQ versus DeepQ on the CartPole environment

We can see that there is very little difference in the training rate of the OPIQ agent and the DeepQ agent, with the OPIQ agent actually outperforming the DeepQ agent near the end of training.

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