Prioritized Exploration in Reinforcement Learning

Di Qi

May 9, 2017

1 Abstract

One identifying challenge of reinforcement learning is the trade-off between exploration and exploitation. To maximize reward, a reinforcement learning must exploit successful actions, but to discover these actions, it must explore actions it has not tried before. In this paper, we focus on the exploration step. [1] In this paper we propose a mechanism for making exploration more efficient by restarting at past states from where we have more to learn, tentatively called *prioritized exploration*. We test prioritized exploration on DeepMind's Atari Deep Q-Learner, a reinforcement learning algorithm that achieved human-level performance across many Atari games. [2] Thus far, we have tested the experiment on the Atari game Breakout and achieved nearly a 50% increase on reward for the same number of steps.

2 Introduction

In reinforcement learning, an agent attempts to achieve some goal by observing and interacting with its environment [1]. At each time iteration, the agent performs actions that affect the underlying environment, and in doing so encounters experiences that lead it to incrementally update its action-selection policy to solve the problem. An experience is defined as a prior observed state, the action taken in that state, the reward received from that action, and the resultant observed state (S_t, A_t, R_t, S_{t+1}). Intuitively, through its experiences the RL learns the state-action pairs that result in high rewards and how to get there [3].

One issue that a reinforcement learning (RL) agent might face in solving this problem is that at each time step, the experiences are not independent and identically distributed, a common assumption in schoastic gradient-based algorithms [1] [3]. Experience replay [4] addresses this problem by saving past experiences and training on a mixture of past and recent experiences, breaking the temporal correlations, and thus stabilizing the algorithm.

This work takes advantage of the non-i.d.d. distribution of the observations to learn new lessons from key past states. Suppose that at a previous state, the agent had a much different action-selection policy than it currently does. Then the agent, were it to encounter the previous state again, would likely take different exploration steps, which could lead to more useful experiences.

In this paper, we investigate how prioritizing which states we train on can make exploration more efficient and effective than if all games were initialized completely randomly. Prioritized exploration allows online RL agents to remember past states and action-selection policies, and restart at states associated with divergent action-selection policies. The key idea is that an RL agent can learn more effectively from some states than from others.

3 Background

Sutton and Barto state "reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem" [1]. The reinforcement learning problem is characterized by an environment and an agent acting within that environment to achieve some goal. The agent partially observes the

environment, collecting some representation of the state S_t of the environment at time t. Importantly, the RL agent does not know the exact reward function. If it did, it would know exactly what actions to take at which states. In value-function based methods for reinforcement learning, the RL agent attempts to estimate the reward function. Specifically, the agent attempts to derive the value of certain states and/or actions. One method of solving the reinforcement learning problem is Q-Learning [5], a value-based method that we use in this experiment.

Given an action-selection policy π, which maps states to actions, and a discount factor $\gamma \in (0, 1)$, we can define an action-value function $q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a] = r(s, a) + \lambda \sum_{s'} p(s' \mid s, a) Q(s', a')$. Note that G_t is defined as the cumulative discounted awards following t and $r(s, a)$ is the reward received by taking action a at the state s. This expresses the derived value for taking the action a at state s. It can be shown that there necessarily exists policies π^* such that

$$q_{\pi^*}(s, a) = r(s, a) + \lambda \sum_{s'} p(s' \mid s, a) \max_{a'} Q(s', a') \tag{1}$$

This is known as the Bellman equation. The same policies will also satisfy the Bellman equation associated with the value function of a state $v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$, namely $v_{\pi^*}(s) = \max_\pi v_\pi(s)$. Then we can express the goal of the RL agent as finding a policy that maximizes the value function at any state s. This reflects the agent's goal of attempting to maximize long-term reward.

Q-learning applies applies incremental estimation to the Bellman equation (1). It incrementally makes a corrections to the action-value function based on new information. The incremental update rule is

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \lambda \max_{a'} Q(s', a') - Q(s, a)) \tag{2}$$

On the right side of the arrow, $Q(s, a)$ is the old value estimation, α is the learning rate, r is the reward received from taking the action a at state s, and λ is the discount factor. We can view $r + \max_a Q(s', a')$ as the learned value of the action a in state s [6]. It can be shown that with sufficient exploration of the action-state space, the algorithm converges to the optimal policy [7].

4 Related Work

In experience replay, the learning agent maintains a collection of its past experiences and repeatedly provides them as input to the learning algorithm, as if the agent experienced again and again what it had previously experienced. One of the reasons for this is that the network will often forget the lessons it learned from past experiences, and thus may need to learn it again [8]. Prioritized experience replay [3] improves upon this by observing that the RL agent may be able to learn more from relearning some transitions than others. The ideal criterion, as they state, would be the amount that the RL agent can expect to learn from that transition in its current state. To measure this indirectly, it ranks importance of transition by the magnitude of the transition's TD error δ, which describes the difference between the estimated value and the realized value. They demonstrate on a simple game that greedily choosing the experience with the highest TD-error to feed into the Q-Learner results in fewer updates needed to achieve the desired value function.

For the implementation, since the TD errors of past experiences do not change over time, and experiences with higher TD-errors are more informative, a binary heap data structure was used to store the ranked experiences. Some state-action pairs might repeat, and with different subsequent rewards and states, but TD-errors were only updated for the transitions that were reused in order to avoid expensive sweeps over the entire experience store.

There were a few problems pointed out in using a purely stochastic approach to prioritized experience replay. Firstly, transitions may have a low TD-error on the initial visit, and thus would not likely be visited, yet could be informative at a subsequent time. Secondly, since errors shrink over time, older high-error transition memories would likely be replayed frequently. This could cause the system to overfit the model,

for example to overemphasizing rare experiences. Lastly, it could be sensitive to a noisy, stochastic reward. The paper introduced two methods of introducing stochasticity, one rank-based and one proportional to the TD-error.

5 Prioritized Exploration

5.1 A Motivating Example

As a motivating example, consider Lucy, who is preparing to take the SAT next month. There are thousands of SAT practice problems that she could work on. This is much too many for her to complete! She has never taken the SAT before, so at first she might simply choose questions at random. Eventually, she will realize that some things she already knows. There is no need for her to do easy algebra problems so often. Intuitively, we can see that Lucy should choose problems that she can benefit the most from. Then how exactly does she determine the problems she can learn the most from? A good indicator is problems that she has missed.

To make the situation more comparable, suppose additionally that upon answering a question, she only knows that her question is correct or incorrect. Therefore, a good test-preparation tactic would be, as she takes practice exams, to revisit problems she has previously missed. She likely missed them because her approach to these problems was incorrect. This likely means that she did not know the answer at all, and was guessing randomly or that she understood the material incorrectly.

However, as Lucy does more and more problems like these, she eventually begins to get the idea for the correct way to approach these problems. If she looks back at the old questions she missed, her thought process is completely different. If she attempts them again with this new mindset, she will learn something different, whether it be the correct answer, or that her thought process is again incorrect. This is much better than selecting problems randomly!

5.2 Method

The goal is to collect more experiences that help with learning the correct action to take at key transitions. In a sense, we want to correct previous mistakes by testing new, better decisions in the same context. Prioritized exploration facilitates this by restarting at previous states where it has more to learn. This us to the main design choices: (1) how to select restarting states, and consequentially, (2) what information to store. In this way we will introduced how prioritized exploration works.

To select restarting states, the ideal criterion is how much can be learned from restarting at a given state. This is not directly accessible, one reasonable proxy that we propose is using the KL-Divergence between the action-selection policy generated by the Q-function for that state as a means of determining how much can be learned. KL-Divergence [9] is a measure of divergence between probability distributions. That the current action-selection policy, a probability distribution for the actions to take at that state, diverges greatly from a prior action-selection policy means that, were we to encounter the same state again, we would likely take different steps. In other words, we would explore a different action in that state, gaining new experiences. To reiterate, we would restart at states associated with action-selection policies that diverge most from the current action-selection policies.

Thus, we store tuples containing the (1) action-selection policy, so that the agent can determine where to restart, the (2) state, from which the agent can restart, and the (3) observation of the state, with which the agent calculates what the current action-selection policy would be. We refer to these tuples as snapshots. It would be important to obtain a sufficiently large and unbiased table of snapshots to learn more effectively. We would also not want to bias towards only a small subset of the tuples, which could lead to overfitting. We would recommend using a sliding window technique, to decrease bias towards a the oldest of past states, which we would expect to have much different action-selection policies due to the lack of training. To further decrease bias, some means of stochasticity also ought to be introduced.

By actively searching for states for which our policy has changed the most, we find opportunities to explore new state-action pairs. The convergence of Q-learning depends upon sufficient exploration of all the possible state-action pairs. Thus, by exploring more efficiently, like prioritized experience replay we can reduce the amount of experience required to learn in exchange for more computation and more memory, which are often cheaper resources than the RL agent's interactions with its environment.

6 Atari Experiments

Algorithm 1: Deep Q-Learning with Experience Replay and Prioritized Exploration

input: number of steps T, experience store capacity N, state store capacity K, state store sampling rate k, start and end probabilities p_{start} and p_{end}, annealing factor τ, saving interval i_s

Initialize experience store EXP-STORE to capacity N
Initialize action-value function Q with random weights
Initialize state store STATE-STORE to capacity K
Initialize probability of restarting p as p_{start}
for *step* $= 1, M$ **do**
  if *Terminal* **then**
    if *probability p* **then**
      Sample k states and of these, restart with state in the STATE-STORE associated with the action-selection policy with the greatest KL-divergence
    else
      Begin a new random game
    end
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  end
  if $p > p_{end}$ **then** $p = p * \tau$;
  if *step* % i_s **then** Add current action-policy, state, and observation to STATE-STORE;
  With probability ϵ select a random action a_t otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
  Execute action a_t in the emulator and observe the reward r_t and the image x_{t+1}
  Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
  Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
  Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from EXP-STORE
  Set

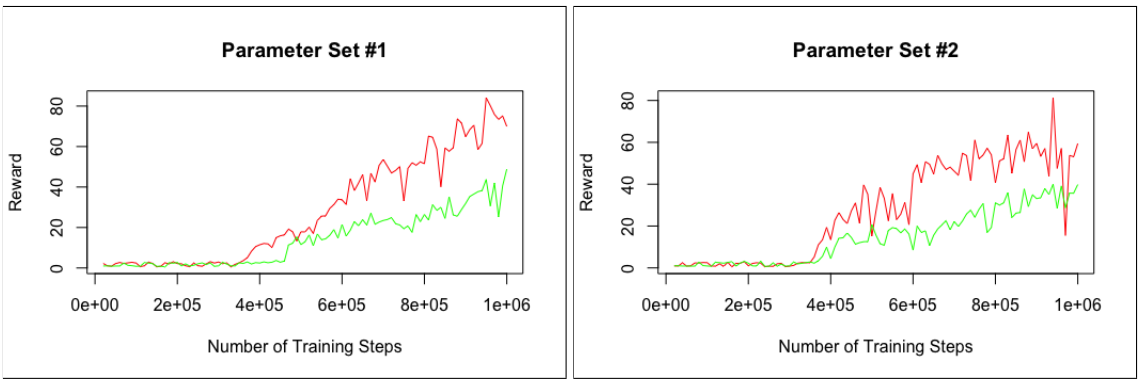
$$y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \lambda \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

  Perform a gradient descent step $(y_j - Q(\phi_j, a_j; \theta)^2$
end

6.1 Implementation

We build off of the code [10] for DeepMind's Atari Q-Learner [2], which uses a convolutional neural network model trained with a variant of Q-learning to learn to play a range of Atari 2600 games. Essentially, the Q function is represented by the neural network. Algorithm 1 shows the algorithm for Q-learning with experience replay in black, and our additions to include prioritized exploration in red. One difference from the pseudocode and the implementation is that the implementation iterates over the total number of steps, rather than the number of episodes and the number of steps in each episode.

We ran experiments for $T = 1,000,000$ steps with varying learning rates. We saved snapshots every 100 steps into a sliding window snapshot table containing $K = 1000$ states. To restart, we used the sampling



(a) Learning Rate 0.0025 (b) Learning Rate 0.005

Figure 1: Interpolation

rate $k = 20\%$, to select a fraction of the snapshot table and determine the snapshot with the greatest KL-divergence to restart from. By selecting a fraction of the total snapshot table with which to restart we introduced some stochasticity. We restarted with an initial probability of $p_{start} = 1.0$, which decreased over time to the end probability $p_{end} = 1.0$ according to the annealing factor $\tau = (p_{start} - p_{end})^{-T/2}$.

6.2 Results

So far, we have run our experiment on the game Breakout. Our results for the learning rates 0.0025 and 0.005 are depicted in Figure 2. We achieved an increase in reward of 43.2% and 49.2% respectively for the same number of steps. This was at the cost of a 15.1% and 16.7% increase in training time respectively. However, these increases in time we expect could be reduced to be negligible with the parallelization of prioritized exploration.

7 Discussion

7.1 Implementing Prioritized Experience Replay

The most important next step to take is to implement prioritized experience replay on DeepMind's Atari Deep Q-Learner. Prioritized experience replay makes learning more efficient by repeating memories that could have more value. Prioritized exploration attempts to make learning more efficient by sampling *new* experiences that we expect to be able to learn more from. It is important to distinguish the two effects to determine exactly how much our algorithm can improve upon the state-of-the-art.

As a preliminary step, we trained a Q-learner with prioritized experience replay to play a simple minesweeper game and compared the results to when we added prioritized exploration and observed a 70% increase in reward for 80,000 steps. This can be seen in Figure 2.

7.2 Generalizability

A subsequent step would be to generalize our experiment. Firstly, we need to revise how we are choosing the states to restart from. Currently, we are restarting at the state with the largest action-selection policy KL-Divergence out of the 20% of the snapshot table that we sample. Analogous to how prioritized experience replay wanted to avoid a lack of diversity and potential overfitting that consistently choosing the past experience with the largest TD-error could cause, we too want to introduce some randomness into how we select the restart state. However, sampling the snapshot table and choosing the most divergent state of these samples is not a very effective method of introducing stochasticity. It fails to consider a large proportion of our data and does not do enough to remove the potential overfitting on a small set of past states. Two methods we are considering are (1) a ranking on the snapshot table, similar to the rankings introduced for

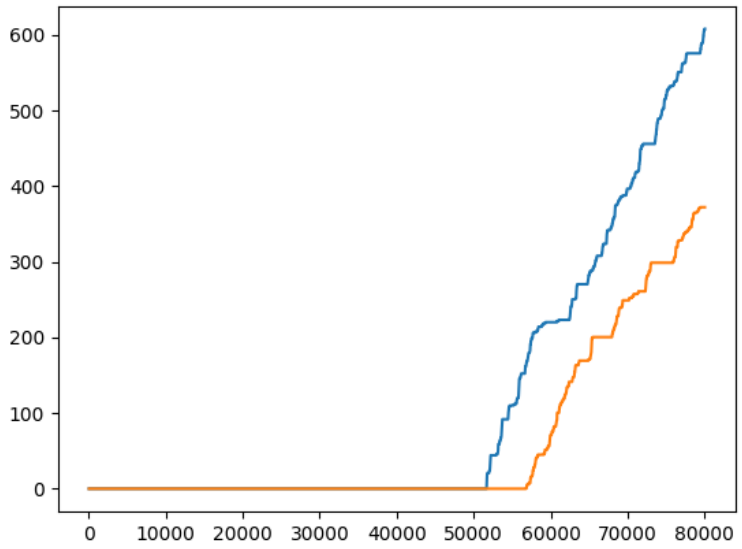


Figure 2: Reward vs. steps plot for Minesweeper Game. Blue is Q-learner with prioritized exploration and prioritized experience replay. Orange is Q-learner with prioritized experience replay but not prioritized exploration.

prioritized experience replay, that is recalculated intermittently and (2) scanning through the table to find the state with the maximum divergence and upon finding a state with a greater divergence than the current maximum, updating the current maximum with only a fixed probability.

Secondly, there are currently 5 important hyperparameters. To reiterate, these are the initial probability of restarting, the final probability of restarting, the number of steps until the end of annealing, the size of the snapshot table, and the proportion of the snapshot table to sample. We need to experiment with these hyperparameters to determine their impact on the stability of this algorithm. Lastly, we need to test prioritized exploration on another Atari games in order to make a stronger conclusion on its generalizability.

7.3 Optimizations

Following this, we would consider tuning the implementation of prioritized exploration. There are many opportunities for parallelism. Most significantly, the calculation of the next restarting state can be done in parallel to the training. Currently, further training is stalled when the program scans through the snapshot table. This optimization would also enable us to enlarge the snapshot table, perhaps obtaining better results. Another opportunity for parallelism would be in the calculation of the the KL-divergences. While currently there is approximately an 18% increase in computation time, we expect that with these optimizations, the increase in computation time would be negligible.

8 Conclusion

This paper introduced prioritized exploration, a method that can make exploration more efficient. We demonstrated in preliminary experiments the effectiveness of this technique, which speeded up learning by almost 50%. We laid out subsequent steps to improve our results and allow us to make stronger conclusions.

I pledge my honour that this paper represents my own work in accordance with University regulations.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[4] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[5] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[6] Mykel J Kochenderfer and Hayley J Davison Reynolds. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[7] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[8] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. PhD thesis, Fujitsu Laboratories Ltd, 1993.

[9] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[10] DeepMind and Ilya Kuzovkin. Deepmind atari deep q learner. `https://github.com/kuz/DeepMind-Atari-Deep-Q-Learner`, 2015.