Improving Exploration in Deep Reinforcement Learning

Abstract—Deep learning has begun to demonstrate great success in reinforcement learning, as deep networks trained using Q-learning have recently shown exceptional performance on a range of Atari video games. However, this approach succeeds only on games for which the major challenge is representational, which can be overcome with deep learning. On many games, performance is still quite poor because the system has no sophisticated means for balancing exploration and exploitation, i.e., deciding when to act in a way that reduces uncertainty about the environment, and when to act in a way that maximises performance. This project looks to develop improved mechanisms of exploration for deep reinforcement learning. The main approach starting from $\epsilon$-greedy methods is to build on existing reinforcement learning methods to improve exploration and find ways to make them scalable in a deep reinforcement learning framework.

I. INTRODUCTION

Deep reinforcement learning has seen recent success with DQN (Deep Q-Networks) [1], initially using simple exploration strategies demonstrating “super-human” performance on many of the Atari 2600 arcade learning environment games [2]. A well-shaped reward signal was the common feature of games for which simple exploration strategies ($\epsilon$-greedy, uniform sampling of replay buffer) were sufficient. While in the case of sparser reward environments the sample complexity can grow exponentially with state space [3] [4], meaning that DQN will not converge on near-optimal Q-value estimates. State of the art performance on the most challenging sparse reward Atari 2600 game, Montezuma’s revenge, has been achieved with intrinsic motivation methods. These methods are attractive as they remain applicable for Reinforcement Learning problems in the absense of the Markov property or the lack of a tabular representation [5], a relevant fit for Deep Reinforcement Learning, they are also known to perform near-optimally for the tabular reinforcement learning case [6]. The idea of intrinsic motivation to encourage an agent to reduce it’s uncertainty was the starting point for this project, both due to this state of the art performance, while also the fact that few others are exploring the same approach. Through use of this method, there is the opportunity to determine whether it can be used more scalably to incentivise an Agent’s exploration. Overall providing the chance to discover what practical improvements can still be made to improve performance of Deep Reinforcement Learning in sparse reward environments.

II. RELATED WORK

This work has been motivated prinicipally by that of Bellamere et al [5] who introduced the notion of a pseudo-count to generalise count-based exploration to non-tabular reinforcement learning. Other count-based approaches in deep reinforcement learning have subsequently been posed such as mapping states to hash codes, which allows counting of their occurrences in a hash table [7]. As such the basis of this work is to continue on the path of exploration with count-based methods. Other approaches such as Variational Information Maximizing Exploration (VIME) [8], an exploration strategy based on maximization of information gain about the agents belief of environment dynamics, have been used to effect on robotic locomotion problems with sparse reward but not generalised to the Atari 2600 environment.

III. COUNT-BASED EXPLORATION WITH DENSITY MODELS

A. Notation

The report assumes a finite-horizon discounted Markov decision process (MDP), defined by $(S, A, P, r, \sigma_0, \gamma, T)$, in which $S$ is the state space, $A$ the action space, $P$ a transition probability distribution, $r : S \times A \rightarrow R$ a reward function, $\sigma_0$ an initial state distribution, $\gamma \in (0, 1]$ a discount factor, and $T$ the horizon. The goal of Reinforcement Learning is to maximize the total expected discounted reward:

$$E_{\pi, P}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$$

Taken over a policy $\pi$, which outputs a distribution over actions given a state.

For the method of Count-Based Exploration let $\rho$ be a density model on finite $S$, and $\rho_n(s)$ the probability assigned by the density model to $s$ after being run on a sequence of states $s_1, \ldots, s_n$, and finally $\rho_n(s) > 0$.

B. Intrinsic Motivation

Intrinsic motivation has been related to Information Gain commonly used to quantify novelty or curiosity and consequently an intrinsic reward [9]. For the deep reinforcement learning problem this has been is approximated through the quantity of Prediction Gain, defined as:

$$PG_n(s) := log \rho_n(s) - log \rho_n(s)$$

For a complete definition of this the reader may refer to Bellamere et al [5].
C. Context tree-switching (CTS)

While more advanced density models have been published with recent success (eg.PixelCNN) [10], for relative simplicity a CTS model [11] was used. CTS is a Bayesian variable order markov model taking a 2D image and assigns to it a probability according to the product of location-dependent L-shaped filters, the prediction of each filter given by a CTS algorithm trained on past images.

D. Tree-Density Model

In practice at run-time we found the computations of a full CTS implementation to take too long for our experimental purposes and so created a lighter-weight version using the core L-shaped context filtering, to run a reduced and faster density model. Importantly retaining features of \( \rho \) being learning positive i.e. \( \rho'_n(s) \geq \rho_n(s) \) for all \( s_1, ..., s_n, s \in S \).

E. Pseudo-Counts

The density model \( \rho \) is used to produce a pseudo-count, which generalises count-based methods to the domain of Deep Reinforcement Learning. Traditional count-based methods have been shown to be near-optimal for tabular Q-learning [6]. The pseudo-count is defined in terms of a density model \( \rho_n(s) \) trained on the sequence of states experienced by the agent.

\[
\hat{N}_n(s) = \frac{\rho_n(s)(1 - \rho'_n(s))}{\rho'_n(s) - \rho_n(s)} = \hat{n}\rho_n(s)
\]

\( \hat{N}_n(s) \) a pseudo-count function generalising the state visitation count function \( N_n(s) \) is computed from the pseudo-count total \( \hat{n} \) and the models re-coding probability \( \rho'_n(s) \), the probability of state \( s \) computed immediately after training on \( s \) [10]. This pseudo-count can be theoretically linked with Prediction Gain (PG) from Section III-B.

\[
\hat{N}_n(s) \approx \frac{1}{e^{PG_n(s)} - 1}
\]

F. Exploration Bonus

Pseudo-counts provided by the density model can then be converted to exploration bonuses \( (R^+_n(s, a)) \), which incentivise the agent to re-visit less frequently visited states and so reduce it’s uncertainty about the environment in the absence of any direct environmental reward.

\[
R^+_n(s, a) := \frac{\beta}{\hat{N}_n(s) + 0.01}
\]

where \( \beta = 0.0001 \) necessarily chosen so that the total episode exploration rewards \( \sum_{n=1}^{T} R^+_n(s, a) \) are not too high and so do not exceed the reward assigned to achieving the environment goal (see section IV). The 0.01 disturbance is used practically simply to prevent \( R^+_n(s, a) \) exploding when \( \hat{N}_n(s) = 0 \).

IV. Environment

In order to run a test bed for count-based exploration bonuses a maze environment, with sparse (single) reward was constructed (see Fig. 1.). With a +1 reward \( (r) \) for reaching the goal, the agent is trained to maximise it’s episode reward and reach the goal. In order to achieve this, due to a sparse reward signal the agent is required to explore the state space accumulating exploration bonuses. These should motivate the agent to eventually reach the goal after sufficient training and propagate the reward associated with completing the maze. So learning Q-value estimates to create an effective policy for solving the environment.

The maze size is 50x50 pixels and the agent is given an episode period \( (T_e) \) of 12500 steps for training. If the agent reaches the goal in less than \( T_e \) steps it is sent back to the start and a new training episode begins, while if the case the agent doesn’t reach the goal in \( T_e \) steps then it begins a new episode from the start. In this way intuitively the agent should be stimulated to find trajectories towards the goal state of the environment. 5 agents are trained on the environment for each each experimental condition, with later plots showing the average and variance of these 5 agents. Each experiment is run over a total training period \( T_l = \{300000, 600000\} \) steps. Finally a basis of DDQN agent was used for all experiments, the idea of the Double Q-learning algorithm, which was introduced in a tabular setting, having been generalized to work with large-scale function approximation [12].

V. IMPROVING EXPLORATION WITH Bonuses

Given the framework for exploration bonuses which has been discussed, the following methods are introduced as the main components to improve the performance of the agent in exploring and solving the environment.

A. Impact of Experience Replay

In tabular reinforcement learning the problem is confined to smaller state spaces and the dynamics can often be handcrafted and modelled with some small discrete MDP (Markov decision
process). In the case of Deep Reinforcement Learning with high dimensional sensory input, we need to train the network on samples of the environment, as the underlying dynamics are unknown and are being approximated. These samples are kept in the ‘Experience Replay’ [13] stored as experiences:

\[ e_t = (s_t, a_t, r_t + R^+_{n}(s_{t+1}, a_t), s_{t+1}) \]

in the replay memory \( D = e_1, ..., e_N \). During learning mini-batch updates are uniformly sampled at random from \( D \). After performing updates to the network, the agent executes an action according to an \( \epsilon \)-greedy policy. The use of an Experience replay is one of the key components of the DQN algorithm [1], allowing for the recent success of using deep neural networks as function approximators in reinforcement learning.

There are practical implications of using an experience replay for training a network, in that, if an input pattern has not been presented for quite a while, the network typically will forget what it has learned for that pattern and thus need to relearn it when that pattern is seen again later. This problem is called the re-learning problem or often referred to as a Q-value forgetting. Choices in the sampling of mini-batches can potentially help to mitigate this problem as discussed in section V-D, while also the size of the replay memory \( D \) is significant to allow for sufficient training on states before they are overwritten in the replay memory.

B. Fresh Counts

In the initial work on using a density model to generate pseudo-counts, it was stated by the authors “...we experimented with dynamic exploration bonuses, where \( R^+_{n}(s, a) \) is computed at replay time, as well as static bonuses where \( R^+_{n}(s, a) \) is computed when the sample is inserted into the replay memory. Although the former yields somewhat improved results, we use the latter here for computational efficiency” [5]. From now on in the report we will term the age of exploration bonuses in the replay memory ‘freshness’, if the bonus is not recomputed at replay time it will referred to as ‘stale’, the longer experiences remains in the replay memory the ‘staler’ they will become. In order to explore the impact of the author’s statement and explore whether time should be invested in making the exploration bonuses dynamic at replay time, experiments were run on the Maze environment and results are discussed in Section VI-A. The hypothesis is that if there is an observable benefit to freshening the bonuses, then can store more fresh experiences in the replay memory, which should give added benefits of less Q-value forgetting while also training on more experiences and reducing over-fitting to the stored experiences.

C. \( n \)-step Q learning

In \( n \)-step Q-learning, \( Q(s, a) \) is updated toward the \( n \)-step return defined with exploration bonuses as:

\[ Q^n(s_t, a_t) = R^+_{n}(s_t, a_t) + \gamma R^+_{n}(s_{t+1}, a_{t+1}) + ... \\
... + \gamma^{n-1} R^+_{n}(s_{t+n-1}, a_{t+n-1}) + max_a \gamma^n Q(s_{t+n}, a) \]

This effectively adds the following \( n \) on-policy rewards and bootstraps the sum of discounted rewards for the rest of the trajectory, off-policy [14]. Resulting in all exploration bonuses directly affecting the values of \( n \) preceding state action pairs, making the process of propagating rewards to relevant state-action pairs potentially much more efficient. Additionally the literature suggests that \( n \)-step methods learn faster than one-step methods on some games [15].

D. Prioritised Experience Replay

A popular framework for sampling the most important experiences from memory is Prioritised Experience Replay. The core idea is to replay important transitions more frequently, and therefore learn more efficiently. In this case the samples from the replay memory \( D \) are prioritised based on TD-error [16]:

\[ \delta(s, a) = (r(s, a) + R^+_{n}(s, a)) + \gamma max_a Q(s', a') - Q(s, a) \]

Therefore training the network on experience’s for which it’s estimations deviate most from observation.

*note:* in practice for this environment the only state where \( r(s, a) \neq 0 \) is the goal state.

E. \( \epsilon \)-Scaling

In DQN the agent proceeds with an \( \epsilon \)-greedy policy after performing Q-value updates. A linear annealing schedule for \( \epsilon \) is common practice in DQN as the agent proceeds through the training period. This \( \epsilon \)-greedy approach is the baseline and was used for initial experiments, though as shown in Section VI-F an \( \epsilon \)-scaling method was later adopted. In creating the \( \epsilon \)-scaling algorithm the idea was that the agent should take more exploratory actions while it’s uncertainty is highest (high \( R^+_{n}(s, a) \)). When \( R^+_{n}(s, a) \geq \text{threshold} \) \( \epsilon \) is boosted to 0.9, then continues to decay each step to a minimum of 0.1 until activated again.

*note* \( \epsilon = [0, 1] \) and represents the probability of taking a random action.

VI. Results

A. Freshening Pseudo-Counts

‘Staleness’ should have a negative effect on Q-value updates from the network as discussed in Section V-B, while the results shown in Figs. 2 & 3, suggest that when the agent performs updates on stale counts the exploration of the environment is greater with regards to total number of distinct states visited. This may be explained by little difference between the 2 methods, as no agent using either fresh or stale exploration bonuses alone is able to solve the environment. This was the first experiment to be run and as such is based on a uniform sampling of the experience replay for a mini-batch size of 32 and 1-step Q update i.e. the traditional DQN setup including exploration bonuses. It should be noted that the experiments for fresh counts were already on this environment considerably more computationally expensive with run times closer to days than hours. The initial results support little benefit from freshening counts given the current mechanism of DQN with exploration bonuses.
Fig. 2. Fresh exploration bonuses sampled uniformly at random from experience memories (Xp) of different sizes (30-300 thousand)

Fig. 3. Stale exploration bonuses sampled uniformly at random from experience memories (Xp) of different sizes (30-300 thousand)
B. n-Step Q Learning

Experiments show a significant improvement in exploration performance of the agent when using n-step returns. Fig. 4, illustrating a step-change in performance with regards to the amount of the maze the agent explores when moving from 1 to n-steps. Based on the literature and experimental performance (eg. run-time) it was decided to fix $n = 100$ [14]. At this point proceeding figures will show episode rewards $r$ for the agent as the goal state in the environment is achieved.

\textit{note}: it became computationally intractable to perform freshening on n-step returns, each mini-batch sample consisting of n times the number of updates.

C. Experience Replay Memory Size

After the insight that n-step Q learning can enable agents to solve the environment, there was still little learning from experience as episode rewards were not consistently achieved by agents, principally due to the mechanisms of Q-value forgetting and slow reward propagation from the goal state. To enable better learning from experience, experiments with larger replay memory different sizes were run shown in Fig. 6. Moving from a memory size greater than the initial value of 30,000 experiences, provides another step change in the agents performance. The relationship is not clear that bigger is better and in fact the best performance of agents with regards to episode rewards is achieved with around 100000 experiences (Fig. 5.). 100,000 experiences providing sufficient memory to be able to train more frequently on states within the buffer and not holding on to ‘stale’ states for too long, balancing over-fitting for too few recent experiences and enough history in the memory. It should be noted that 100,000 states is more than enough to cover the entire state-space of $\approx 8000$ states. Therefore with prior knowledge this would be enough memory to converge on a near optimal solution and have no Q-value forgetting though experimentally a much larger memory is needed. This point suggests there is significant room for innovation around compression of the replay memory size while keeping realistic large-scale performance, further work (Section VIII) will explore the sample-based model literature to produce a similarity metric between states which is computationally cheap to compress the experience memory.

D. Mini-Batch size

Mini-batches refer to the sample size taken from the replay memory. The literature tends to use at a fixed 32 samples [1], while experimentally results also show that with improvements provided from n-step Q learning and a larger replay memory, increasing the mini-batch size for uniform random sampling has no benefit on performance Fig. 7. A smaller uniformly sampled batch therefore seems to perform the best.

E. Prioritised Experience Replay

Sampling of the experience memory based on TD-error as outlined in Section V-D was implemented, though with little improvement over uniform sampling. Of note is the result that prioritised experience replay achieves best performance with smaller memory sizes. Fig. 8, as desirable attribute when moving to much higher dimensional state spaces and indicative of the improved performance of the method on Atari 2600 games.

F. $\epsilon$-scaling

The performance of this agent seems to provide an incremental improvement over a basic $\epsilon$-greedy approach in DQN, with no quantifiable computational cost. Initial results in Fig. 9 suggest that $\epsilon$-scaling is beneficial in this environment but it would be useful to see how this improvement translates to other higher dimensional state spaces.

VII. CONCLUSIONS

This report has presented a study of count-based methods for exploration in deep reinforcement learning, giving insights that were not provided in the literature. Principally that prioritised experience replay seems to provide no quantifiable improvement to an agent with count-based exploration bonuses. Presenting an opportunity to develop new methods of shrinking the replay memory size while preserving important experiences. This would improve the performance of the exploring agent, by improving the quality and distribution of experiences which it uses to learn Q-value estimates. Finally the methods used, have far improved performance in the constructed sparse reward environment than the previous gold standard in Deep reinforcement learning (DQN).

VIII. FUTURE WORK

This work should continue as described in the conclusions, towards smarter storage of experience in replay memory. A promising avenue to achieve this comes from existing reinforcement learning research in proto-value functions [17]. Proto-value functions are constructed using the eigen-functions of the (graph or manifold) Laplacian, which can be viewed as undertaking a Fourier analysis on the state space graph. The use of a graph representation of replay memory in deep reinforcement learning is therefore yet to be exploited.
Fig. 4. DDQN agent with exploration bonuses, improvements using n-step Q returns

Fig. 5. The per episode rewards for agents of different experience memory size
Fig. 6. The states visited graph for agents of different experience memory size

Fig. 7. Results of varying mini-batch size for uniform random sampling of the replay memory (300,000) and 100-step Q updates
Fig. 8. Impact of prioritised sampling of memory with stale bonuses

Fig. 9. Comparison of the performance of 3 different Agents, 1. Minimal DQN Agent (DQN), 2. DQN Agent with count-based exploration bonus (Count), and 3. Agent with exploration bonuses using epsilon scaling (Epsilon Scaling). All agents are using 10 step Q returns, a strong motivation in the least to using exploration bonuses.
REFERENCES