Brief Survey of Count-Based Exploration

Jonathan Campbell
School of Computer Science
McGill University, Montréal
Québec, Canada

1. INTRODUCTION

The problem of exploration is a central issue in reinforcement learning. Only by exploring the state space of an environment sufficiently can an agent discover the optimal policy. Simple exploration strategies exist and are widely used, but are sometimes insufficient, particularly in environments with noisy and/or sparse rewards. In this report we briefly describe more sophisticated, 'count-based' exploration strategies. The count-based approach relies on keeping visit counts for each state and artificially boosting the action-values of less-visited states, in order to encourage revisiting of those states to get a better estimate of their action-values.

We first present simple exploration strategies (that can be used in conjunction with the count-based approach), and then describe some count-based approaches found in the literature. Performance of these latter approaches will then be evaluated on a suitable novel environment.

2. RELATED WORK

Several non-count based approaches exist for exploration, some being model-based algorithms (including E^3, R-Max, Model-Based Interval Estimation (MBIE) and MBIE with Exploration Bonuses), while other, more recent approaches have been motivated by deep reinforcement learning.

E^3, or Explicit Explore or Exploit, works with two policies: an exploration policy and exploitation policy. The exploration policy favors reaching 'unknown states' – states for which the agent has not yet visited a certain number of times. Visiting known states (states visited above the threshold) with this policy yields zero reward, while visiting the unknown states receive a positive reward [4].

The R-Max algorithm follows a greedy policy, but changes the update rule for a state-action pair depending on the number of visits to that pair. If the state-action pair has been visited at least m times, then the action-value update follows the regular Bellman model-based backup. If it has not been visited m times, the action-value is instead set to $\frac{1}{1-\gamma}$, where γ is the discount factor [3]. This value for less-visited states can be thought of as an exploration bonus which encourages the agent to visit these states until a certain visit count has been reached.

MBIE explicitly forms a model of the environment and maintains confidence intervals for each state-action pair, both for the mean reward of the state-action pair and for its transition probability distribution. An exploration bonus is given to states based on the probability that they will yield a high reward [12]. MBIE with Exploration Bonuses uses the same general idea but adds a bonus of the form $\frac{\beta}{\sqrt{\#(s,a)}}$ to an action-value during updates [9].

While these model-based approaches are useful in certain situations, in larger state spaces it becomes less feasible to form a model of the environment due to the large number of states.

Other forms of exploration have been motivated by deep reinforcement learning. Lipton et al. had success using Thompson sampling to choose actions [5]. Stadie et al. assign exploration bonuses from a concurrently-learned model of the system dynamics (the more novel a state, the higher the bonus) [7].

3. SIMPLE EXPLORATION STRATEGIES

Simple, common techniques for exploration include biased selection of actions as well as optimistic initialization of action-values.

Action selection methods are perhaps the most simple form of exploration. The idea here is to select a non-optimal action with some probability. The basic form of this strategy is referred to as ε-greedy exploration, which is to select a random action with probability ε and the greedy action with probability 1 - ε. Another form called softmax action selection gives all actions a certain probability corresponding to their action-value estimate, with better actions getting a higher probability than worse actions. An example of such a probability distribution to be used is the Boltzmann distribution, which selects an action a in state s as follows:

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_b e^{Q(s,b)/\tau}}$$

The advantage of these approaches lies in their simplicity. Many algorithms can be configured to use this approach for action selection, and they well in both discrete and continuous state spaces.

Another common technique is to optimistically initialize all action-values. By artificially inflating all action-values beyond their expected true values, the learning agent is encouraged to try all actions at least a certain number of times before a true action-value dominates the optimistic initialization. This trick can be performed whether or not ε-greedy/softmax is used since even choosing a greedy action will let the agent choose an artificially-inflated action value.

4. COUNT-BASED METHODS

Sometimes simple exploration as described above is not enough, and so more advanced methods like count-based exploration are needed. The idea of count-based exploration is to direct exploration into lesser-visited states, usually by applying an exploration bonus to the observed reward during each state transition. Such a strategy requires maintaining visit counts for each state (or sometimes state-action pair); there are thus different approaches for discrete vs. continuous (or intractably large) state spaces.

4.1 Discrete state spaces

In the discrete state space case, one algorithm that incorporates exploration bonuses is Delayed Q-Learning with Interval Estimation (DQL-IE), an algorithm similar to regular Q-Learning but which incorporates some features from the model-based MBIE-EB [8].

In DQL-IE, an action-value update will only take place if the difference between the current Q(s, a) value and a running average of failed updates to Q(s, a) is larger than ε. If the update is successful, then the running average for that (s, a) pair is reset to 0 and an exploration bonus is added to the updated action-value of the following form:

$$r^{+}(s,a) = \frac{\beta}{\sqrt{\#(s,a)}}$$

where β is a hyperparameter and #(s, a) is the current count of observances of state s and action a (the count is reset to zero when an update is performed).

If the state-action pair has been experienced only a few times, then the term will be large (depending on the value of beta) and so the action-value will be artificially inflated encouraging the agent to revisit that state-action pair in the future. In the limit, as the number of visits goes to infinity, the term will go to zero and thus the true action-value will become dominant.

The difficulty with this approach lies in the fact that a state-action pair must first be observed before the exploration bonus can be added to it. However, in a discrete case with a tractable state space, it is usually not an issue to visit states at least once. In fact, the DQL-IE algorithm was proved to be PAC-MDP, meaning that it can achieve near-optimal performance with high probability [8].

4.2 Continuous/large state spaces

Sometimes it is not practical to keep track of state visit counts, either because the state space is continuous or because it is so large that a state may never be visited more than once. In such cases, states must be mapped to a lower-dimensional space in order for visit counts to continue being meaningful. Two methods will be reviewed here: the use of a hash function to discretize states, as well as the use of density model.

4.2.1 Hashing

Tang et al. presented a method for count-based exploration in a high-dimensional state space which uses a hash function to discretize states [10]. Visit counts per hash are maintained in order to add an exploration bonus to the observed reward of the form:

$$r^{+}(s,a) = \frac{\beta}{\sqrt{\#(\phi(s))}}$$

where β is a hyperparameter and ϕ is the state hashing function. Unlike DQL-IE, here only (hashed) state visit counts are maintained and not state-action counts, since using the latter showed no significant performance improvement [10].

The choice of hash function can be decided using prior domain knowledge. Any hash function should group together like states and seperate dissimilar ones. One simple hashing algorithm suggested in their paper is SimHash, which uses angular distance to measure similarity, outputting a k-dimensional vector as hash for a state s as follows: $\phi(s) = \operatorname{sgn}(Ag(s)) \in \{-1,1\}^{k}$, where g is an optional state pre-processing function and A is a matrix filled with values drawn from a Gaussian distribution with mean 0 and standard deviation 1. The value of k controls the length of the hash; the higher the value, the fewer the number of states per hash.

In another formulation, the authors propose the use of an autoencoder to learn good hashes, which performs better when dealing with image data.

4.2.2 Density models

Another way to deal with intractable state spaces is to use a density model over the state space, an approach suggested by Bellemare et al [1]. Such a model will take in a state and output its likelihood (the probability of its observance). It can also be trained on a particular state; thus, the difference in likelihood of a state directly before and after training on it can be used to determine a pseudo state-count. This pseudo-count can then be used to generate an exploration bonus and apply it to the reward in the same fashion as described previously.

In particular, the probability of a state x after having trained the model on x is called the 'recoding probability', $p'(x)$. With the recoding probability and original probability $p(x)$, we can obtain a pseudo-count using the following formula:

$$\widehat{N}(x) = \frac{p(x)\,(1-p'(x))}{p'(x)-p(x)}$$

The choice of density model, akin to the hash function case, can be inspired by prior domain knowledge. One model suggested in the paper by Bellemare et al. is the CTS (context-tree switching) model. The CTS model splits a state into factors and outputs a probability for the state by multiplying the probability of its factors [1]. For example, if the state is an image (grid of pixels), each pixel could correspond to one factor. Each factor is modelled seperately, with its probability conditioned by some state context. This context could be empty (i.e., the marginal distribution), the neighboring pixels, or neighbors in the form of an L shape around the current factor (which performed well in their experiments).

Other density models include the PixelCNN architecture, which is tuned for better performance on images [11]. Both models have shown great improvement when run in conjunction with deep reinforcement learning agents on Montezuma's Revenge, a game infamous for the amount of exploration needed to solve it.

5. RESULTS & DISCUSSION

5.1 Environment

The environment used to test the described exploration strategies is a novel one being worked on as part of ongoing thesis work. It is based on the popular roguelike game NetHack [6]. Each episode takes place in a small 8x8 grid, with the agent (player character) and a monster (a giant bat) each taking up a random position on the grid.

The objective of the agent is to destroy the monster. At each step in the episode the agent can take one of several actions: move one step closer to the monster, attack the monster with a close-distance or ranged weapon, equip a weapon, and so on. There are several ways to accomplish the goal. The agent can simply approach the monster and then attack it with its bare fists; however, this approach is dangerous since the monster could then fight back. The agent could first decide to equip a ranged weapon (the bow), and then approach and attack with the bow to do slightly more damage, but again it is dangerous. The optimal approach is equip the bow as soon as possible, move such that the monster is in the agent's line of fire (a straight or diagonal line between the agent and monster) and then fire arrows using the bow continuously until the monster dies. This approach will accomplish the goal in the safest way possible and thus obtain the highest average reward.

Rewards in the environment are sparse, only given at the end of an episode (a positive reward for monster death, or negative reward for player death), making exploration important. Further, an episode can have more than 20 steps, and there are many actions to choose from at each step (approach monster, attack monster at close range, attack monster at far range, line up with monster, equip bow, among others), and in addition, as shown above, there can be several ways to get a positive reward, but many are less safe/consistent in the long run. Thus it is hoped that directed exploration of the form described above will help the agent converge to the optimal policy.

The state space is hand-configured and comprises several pieces of information about the game state. This includes the items in the player's inventory; the items equipped by the player; the monster the player is currently fighting; the player's health; whether the monster is in the line of fire with the player; the distance between player and monster, and other things not listed here. The state space is discrete except for two quantities: the player's current health, and distance from player to monster, which are both normalized to the [0, 1] range. When testing the discrete algorithms (e.g., DQL-IE) on this environment, these two quantities are each discretized into a one-hot encoded vector.

Experiments run with the hashed state-counts use the SimHash algorithm. Experiments with the density model use the CTS model with a specially-configured state factorization. In particular, each part of the state space (equipped items, monsters present, etc.) uses its own factor which is conditioned on different information. For example, the factors that represents the current monster and the player's current inventory are conditioned on nothing (since they are given at the start of an episode). Meanwhile, the factor for the currently-equipped items is conditioned on all state information (since, e.g., the choice of equipment may be influenced by the monster the agent is currently facing, or the agent's current health total, or whether the monster is in the line of fire with the agent, and so on). In total, each state comprises 14 factors (with one model for each factor).

5.2 Methodology

Each algorithm was implemented and tested on the aforementioned environment. Delayed Q-Learning with Interval Estimation was implemented and compared to regular Q-Learning in the tabular case. The two different count-based forms were implemented and tested with both Q-Learning and Sarsa with linear function approximation, and compared to regular Q-Learning and Sarsa with no exploration bonus. The density model factors were handcrafted for the environment as described above but the implementation uses code sourced online for the Context-Tree Switching model [2].

The algorithms that used Q-Learning used an ε-greedy action selection with the value of ε being linearly annealed from 0.25 to 0.05 through the course of learning. The Sarsa algorithms followed a greedy policy throughout.

For each algorithm, a short grid search was run to determine the best values for each hyperparameter. Results are reported using the best hyperparameters for each model.

Each algorithm was run for 3000 episodes with a maximum of 100 steps per episode. To evaluate the policy of an algorithm, at 10 equally-spaced points during the 3000-episode run, 10 episodes were run using the current action-values/weights of the algorithm, and results (total reward per episode) were averaged over these 10 episodes. The final result presented in these charts is the average of these total rewards per episode, i.e., the average total reward throughout the learning process.

Only observed rewards are included in the results; any exploration bonus is omitted from the total reward per episode since it would conflate the true results.

5.3 Results

Results for the exploration strategies are presented in figure 1 for the discrete (tabular) case and figure 2 for the continuous case (using linear function approximation).

As seen in figure 1, DQL-IE performed significantly better than regular Q-Learning. The result for the regular Delayed Q-Learning algorithm is also presented (this algorithm is similar to DQL-IE, but less practical and has a weaker form of exploration bonus).

For the algorithms with linear function approximation shown in figure 2, regular Q-Learning again did poorly, followed by Q-Learning with density models and with state hashing, but the difference in reward here is not as significant as the tabular case. The same progression is shown for linear Sarsa, which has the regular Sarsa algorithm doing the worst, followed by linear Sarsa using density models and then with state hashing. It is possible that the handcrafted density model factors cause this lower performance; the actual density model selected (CTS) may also not be suitable for the state space of the environment. (Sarsa had higher performance probably since it followed a greedy policy.)

Due to time constraints, only 3000 episodes were run for each algorithm, preventing them from finding the optimal policy. The results are therefore incomplete since most of the advantages of directed exploration would likely only be shown later on in the learning process. However, the result still shows that the algorithms with exploration bonuses do perform better than their regular counterparts in this short amount of learning time.

6. CONCLUSION

This report demonstrated the utility and means of count-based exploration strategies with regards to complex environments, particularly when dealing with large state spaces. Approaches for both discrete and continuous/large state spaces were summarized, namely Delayed Q-Learning with Interval Estimation and exploration bonuses through state hashing and the use of a density model. Each algorithm was implemented and tested on an environment where exploration is important.

7. REFERENCES

[1] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Neural Information Processing Systems*, pages 1471–1479. 2016.

[2] M. G. Bellemare, J. Veness, and E. Talvitie. Skip context tree switching – reference implementation. `https://github.com/mgbellemare/SkipCTS`, 2014.

[3] R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, Mar. 2003.

[4] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Journal of Machine Learning Research*, 49(2-3):209–232, Nov. 2002.

[5] Z. C. Lipton, J. Gao, L. Li, X. Li, F. Ahmed, and L. Deng. Efficient exploration for dialog policy learning with deep BBQ networks & replay buffer spiking. In *Neural Information Processing Systems*, Deep Reinforcement Learning Workshop, 2016.

[6] NetHack Dev Team. NetHack 3.6.0: Download the source. `http://www.nethack.org/v360/download-src.html`, 1987–2015.

[7] B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. In *Neural Information Processing Systems*, Deep Reinforcement Learning Workshop, 2015.

[8] A. Strehl. *Probably Approximately Correct (PAC) Exploration in Reinforcement Learning*. PhD thesis, Rutgers University, 2007.

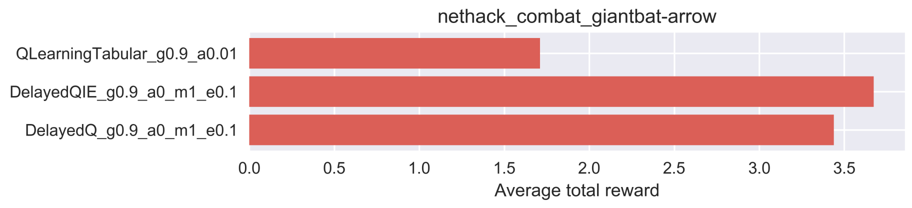

Figure 1: Performance of Delayed Q-Learning with Interval Estimation (DQL-IE) and tabular Q-Learning on the discrete NetHack combat environment. Best-performing hyperparameters for each model are included next to their names.

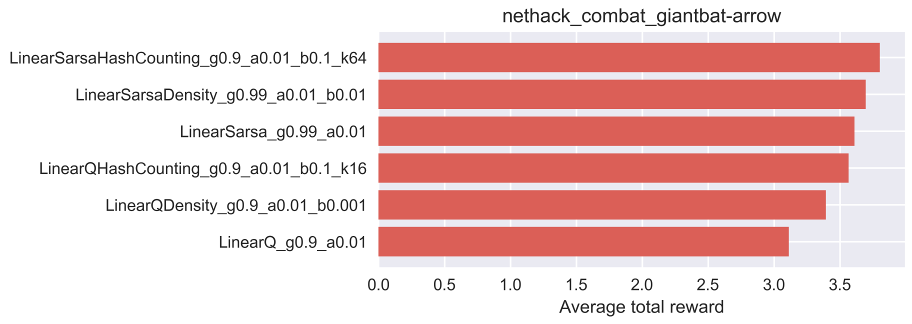

Figure 2: Performance of hashed state and density model exploration bonuses using both Q-Learning and Sarsa with linear function approximation on the continuous NetHack combat environment. Best-performing hyperparameters for each model are included next to their names.

[9] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, Dec. 2008.

[10] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel. #Exploration: A study of count-based exploration for deep reinforcement learning. In *Neural Information Processing Systems*, Deep Reinforcement Learning Workshop, 2016.

[11] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *Journal of Machine Learning Research*, 48, 2016.

[12] M. Wiering. *Explorations in Efficient Reinforcement Learning*. PhD thesis, University of Amsterdam, 1999.