Using Bootstrapped DQN as an exploration methodology in Rainbow Deep Reinforcement Learning

Shane Fearon
Queen's University Belfast, EEECS

Abstract—**Improvements to the Reinforcement Learning algorithm DQN have been developed and successfully combined in a previous study. Yet there are still extensions which have not been tested in this configuration.**
This study will investigate the influence that using Bootstrapped DQN will have when combined with the Rainbow algorithm, in place of Noisy DQN, as an exploration methodology. We will be using the OpenAI gym CartPole-v0, Acrobot-v1, and NChain testing environments to study the effects.

Index Terms—**Reinforcement Learning, Rainbow (DeepMind), Deep Q-Networks, Bootstrapped DQN**

I. Introduction

REINFORCEMENT LEARNING (RL) is a machine learning technique which gives an agent the tools needed to learn in an interactive environment. The agent learns not by being given correct inputs and outputs as in supervised and unsupervised learning, but by its own experimentation with the environment, based on trial and error. Numerous methods have been developed to improve learning times, increase performance and exploration of actions in RL. One example of a RL methodology is the Deep Q-Network algorithm (DQN), which was popularised in recent years when it was shown that a machine using DQN was able to learn from a raw pixel input how to play Atari games with the same skill as a human [1].

Many extensions to DQN have been developed which improve the performance and stability of the algorithm. The Rainbow algorithm was developed by taking 6 of these extensions (which solve radically different issues) and integrating them with DQN. This technique worked well for these 6 algorithms, but this was not an exhaustive list of compatible extensions, which raised the possibility of combining more extensions with the Rainbow algorithm [2].

We aim to show how well Bootstrapped DQN, an exploration extension of DQN, works when integrated into Rainbow. We will be looking at how it affects performance and exploration when compared to the original Rainbow program.

We will approach the problem by first replicating the Rainbow algorithm using the Classic Control testing environment, so that we obtain a benchmark to compare against our modified Rainbow algorithm using Bootstrapped DQN. We will then integrate Bootstrapped DQN the new program, replacing Noisy Nets. We shall call this modified version of the Rainbow algorithm which uses Bootstrapped DQN "Bootstrapped-Rainbow". We will test our Bootstrapped-Rainbow program to show the impact on performance when compared to Rainbow, and to Rainbow with no Noisy DQN. This configuration of DQN extensions has not been tested together before and should provide us with informative results.

We will be using Python to develop our software as it has widespread support within the RL community. We will begin by setting up our version of the Rainbow algorithm, and will test it using the Classic Control environment. This environment was chosen in order to receive our results promptly; the Atari Learning Environment, as used in the original Rainbow paper, takes too long to train for our testing purposes, roughly 10 days per game. We will then receive the results of training and evaluating the Rainbow algorithm in this environment. We will then remove Noisy DQN and repeat the experiment to obtain results for this variant. Bootstrapped DQN will then be integrated into our program to create our Bootstrapped-Rainbow program, and the experiment will be repeated using the same environment as before, to receive our results to compare against Rainbow, and Rainbow with no Noisy DQN's results.

Our experiments showed that Bootstrapped-Rainbow performs much better than Rainbow in certain environments, and worse in other environments, due to the nature of exploration vs exploitation.

II. Literature Review

A. Temporal Difference Learning

Temporal Difference Learning (TD), a model-free policy prediction method, was made popular by Suttons paper (1988) when he provided proof of its convergence to an optimal policy for special cases and related it to supervised learning [3]. Sutton comprehensively defines TD as a technique which utilises bootstrapping to adjust the value function for states, over all time-steps in the case of TD(λ), in contrast of Monte-Carlo prediction, which required termination of episodes before it can adjust the value functions. Perhaps TDs most prominent role was when Tesauro created TD-Gammon in 1992, the first computer to learn the game of backgammon model-free and achieve a level of play just below a human master by being trained by the TD(λ) algorithm [4]. Tesauro concludes by noting that TD is a promising algorithm for general purpose

Shane Fearon is with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, UK, e-mail: sfearon08@qub.ac.uk

policy prediction/control, however, in Tsitsiklis and Van Roys Paper, they showed that divergence can occur in TD when using a nonlinear function approximator [5].

B. Q-Learning

Q-Learning is a model-free off-policy prediction/control method which uses actions and values to determine the quality of those actions, instead of being based purely on a value function-based system and allow the policy to influence its trajectory through a state space. Q-Learning adjusts two policies at once, target and behaviour, so the RL Agent selects the appropriate action for a state by adjusting the behaviour policy toward the target policy to end up in states with the highest Q-Function value (similar to the value function but considers the actions available at states also). Watkins and Dayan define this method and provide proof of its convergence when all actions in all states are repeatedly sampled and action-values are represented discretely [6].

C. DQN

An issue that faces Q-Learning is that of scalability. Using a Q-table lookup for a state and action value is a naïve way of storing and looking up the Q-values for a state-action, as the table increases in size quickly with addition of new states and actions to become an intractable table to work with. To tackle this, DQNs use a non-linear neural network function approximator (Q-network) to approximate these Q-Values. This approach was used by DeepMind in their Playing Atari paper (2013), which provided impressive results, even though a formal proof of policy convergence was not guaranteed [1], [7]. Rummary and Niranjan prove the importance and efficiency of on-line updating in Q-Learning in continuous state spaces, converging in fewer updates than previous methods [8].

D. Experience Replay

Another problem with DQN is that updating the Q-values based on the last n time-steps leads to the Agent updating its policy based on highly-correlated data, leading to a policy that is good for those correlated experiences, but not good for other areas not correlated to the currently sampled experiences. To address this, an Experience Replay Buffer is used. When an Agent transitions from one state to another, that experience is stored in a buffer, and the policy is updated at intervals by taking random samples from this buffer, which are more independent, and assists in convergence to the optimal policy. Even experience replay has its issues. Since some experiences are more important than others when updating the policy, with vanilla Experience Replay, there is the possibility that important experiences are being overwritten by less important experiences, coined as Catastrophic Forgetting.

Prioritised Experience Replay is a measure used to avoid this problem and is studied by Isele and Cosgun [9]. In this paper, they explore 4 different strategies for selecting which experiences should be stored in the experience buffer and avoiding Catastrophic Forgetting. Their approach proves very useful for DQNs and was used in the Playing Atari paper [1].

E. Generative Adversarial Networks

GANs provide a way of generating data which may otherwise be expensive to gather. As in Goodfellow (et al.)'s paper on the subject, a GAN consists of two models, a Generator and a Discriminator. The Generator aims to create data, and the Discriminator aims to sniff out whether the data was generated or is real until the generator produces an indistinguishable data set [10]. This method can prove useful for Experience Replay, as seen in Huang et al.s paper on generating good quality experiences by having the Generator and Discriminator both improve during the two-player game (EGAN), and was used to pre-train an RL agent for faster learning [11]. GANs can also be used in-place of DQNs, as they are capable of generating an optimal Q-value distribution, as shown in Lyle(et al.)s paper GAN Q-Learning [12]. This innovation has still not addressed the issue of providing a proof of convergence, and is also volatile at times, but is still a viable alternative to DQNs. Another paper by Pfau and Vinyals discusses the possibility of viewing GANs as Actor-Critic models, but failed to show any implementation, only discussing at high-level how the two methods could work together and is therefore not useful for our scenario [13].

F. Bootstrapped DQN

Efficient exploration of the state space is one of the areas in which DQNs could be improved. Bootstrapped DQN aims to have deep exploration and deep learning. It plans to not only find the best reward, but also the best future information, since informative actions may lead to higher rewards. It does this by addressing the ϵ-greedy dithering that is used in DQNs for exploration of the state-space [14]. Bootstrapped DQN provides reasonable uncertainty estimates for neural networks for efficient and deep exploration. This paper provides us with a very useful algorithm for deep exploration.

G. Rainbow

The Rainbow paper studies the combination of 6 different extensions to DQNs and the integration of them into a single algorithm [2]. Extensions used include Double DQN, Prioritised experience replay, Duelling networks, the multi-step bootstrap targets extension A3C, Distributional Q-Learning [15] and Noisy DQN. Rainbow has had excellent results from this combination of complimentary extensions. In our approach, we will be investigating the use of Bootstrapped DQN, as it is suggested in the Rainbow paper, and should provide deeper exploration than Noisy DQN, which is currently in place in Rainbow.

H. Noisy Networks (Noisy DQN)

The Noisy Networks paper introduces a general method for exploring the state-space in deep reinforcement learning which does not require any sort of hyper-parameter tuning [16]. It does this by injecting noise into the This methods of exploration is in place in Rainbow, and is what we wish to replace with Bootstrapped DQN in this paper.

III. Background

This section provides a mathematical background on Markov Decision Processes and Deep Q-Networks

A. *Environments, Agents, and Exploration*

At each discrete time step in our environment $t = 0, 1, 2...$, the environment provides the agent with an observation S_t, which the agent uses in its selection process for choosing an action A_t to take, and then the environment provides the next reward R_{t+1}, discount γ_{t+1}, and state S_{t+1}. This interaction is called a Markov Decision Process, or MDP, which can be represented by a tuple $< S, A, T, r, \gamma >$, where S is a finite set of states, A is a finite set of actions, T is the transition function, r is the reward function, and γ is a discount factor. Where:

$$T(s, a, s') = P[S_{t+1} = s' | S_t = s, A_t = a]$$

$$r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

$$\gamma \in [0, 1]$$

The MDPs in our experiment will be episodic with a constant discount factor $\gamma_t = \gamma$, except on episode termination where $\gamma = 0$. For our agent, the selection of an action is based on a policy π, which defines a probability distribution over actions for each state. From the state S_t, the expected discounted return is the discounted sum of future rewards collected by the agent, defined as $G_t = \sum_{k=0}^{\infty} \gamma_t^{(k)} R_{t+k+1}$, where the discount for a reward k steps into the future is the product of the discounts encountered before that time, given by $\gamma_t^{(k)} = \prod_{i=1}^{k} \gamma_{t+i}$. The agent's goal is to maximise the expected discounted return by finding a good policy.

This policy can be constructed as a function of other learned quantities, as in value-based reinforcement learning, where the agent calculates an estimate of the expected discounted return G_t, which is called "value", when it follows a policy π starting from a given state s, given by $v^{\pi}(s) = \mathbb{E}[G_t | S_t = s]$ or state-action pair (s, a), given by $v^{\pi}(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$.

ϵ-greedy is a common way to explore the state-action space and find new policies. This involves acting "greedily" and picking the action that has the highest value, with probability $(1 - \epsilon)$, and otherwise pick a random action, with probability ϵ, with the hope that picking actions with a lower value at a state S_t may lead to greater total value overall, and not to become stuck in a local minimum.

This method of exploration is fairly simplistic, and other methodologies exist for more efficient and deeper exploration. Bootstrapped DQN is one such example, and is the methodologies we will be examining for use in Rainbow.

B. *Deep Q-Networks*

Deep Q-Networks (DQN) were first demonstrated in 2015 [7], combining deep neural networks and reinforcement learning to play Atari games at a super-human level. A convolutional neural network was used to pre-process video frames from Atari games, which provided the DQN with a state S_t. At each step, the agent selected an action ϵ-greedily with respect to the action values based on the current state S_t. Then, a transition tuple was appended to a replay memory buffer that holds the last million transitions, where a transition is defined by $(S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1})$. The parameters of the neural network were then optimised by using stochastic gradient descent to minimise the loss, defined by:

$$(R_{t+1} + \gamma_{t+1} max q_{\bar{\theta}}(S_{t+1}, a') - q_{\theta}(S_t, A_t))^2$$

where t is a time step randomly selected from the replay memory buffer. The gradient of the loss is only back-propagated into the parameters of the online network θ, which is used to select actions, and $\bar{\theta}$ is the *target network*, which is a copy of the online network which is updated at regular intervals, and is not optimised. The optimisation was performed using RMSProp, a variant of stochastic gradient descent, on mini-batches taken uniformly from the replay memory buffer. So in the loss equation above, t is a random time index from the last million transitions stored, instead of the current time.

IV. Component Description

A. *Rainbow*

The Rainbow paper [2] examined 6 different complimentary extensions to the DQN algorithm. Each extension solves a different issue of DQNs to improve performance and increase exploration:

1) **Double DQN** - used to address a bias overestimate of Q-Learning
2) **Prioritised Experience Replay** - to avoid sampling correlated data samples
3) **Duelling Networks** - for state and action value separation, so good actions are not hindered by bad states
4) **A3C** - for diverse experience sampling
5) **Distributional Q-learning** - to mitigate the effects of learning from a non-stationary policy
6) **Noisy DQN** - to introduce uncertainty for better exploration

The Atari Learning Environment along with ablation studies were both used to show the contribution of each of these extensions. Noisy DQN was used to improve exploration, as the authors noted that in some games in the Atari Learning Environment, many actions were needed in order to obtain the first reward.

B. *Noisy DQN*

Noisy DQN uses a noisy linear layer in a neural network which is used to produce the probability distribution over the actions for a given state [16]. It uses a deterministic and noisy stream, then combines the two using the following:

$$y = (b + Wx) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^w)x)$$

where ϵ^b and ϵ^w are random variables, and $\odot$ represents the element-wise product. This is one of many possible methods to choose from to improve exploration during training.

C. Bootstrapped DQN

Bootstrapped DQN is an extension to DQNs that facilitates deeper exploration of sub-optimal actions in order to increase overall value. It does this by having k different ”heads”, each of which has its own Q-value function, in our case, each head is its own neural network. At the start of each episode, the agent selects one of these heads and follows the policy which is optimal for that head for the duration of the episode. The algorithm then trains a random subset of the heads using the resulting data for that episode. This leads to each head having different behaviour, and facilitates deeper exploration than ϵ-greedy.

D. The Combined Agent

In Rainbow, Noisy DQN is used alongside ϵ-greedy to introduce uncertainty so that the agent may explore the state space more than if it were using ϵ-greedy by itself. Our agent will not include Noisy DQN, but will instead use Bootstrapped DQN in its place, alongside ϵ-greedy. This will affect how our agent explores the state space, leading to deeper exploration than using Noisy Nets.

V. Experiment Description

We now describe the methods and setup used to evaluate our Rainbow agent, a variant of Rainbow with no Noisy DQN or Bootstrapped DQN, and our Bootstrapped-Rainbow agent.

We evaluated all 3 of our agents using 2 Classic Control environments (Cartpole-v0 and Acrobot-v1) and 1 Toy Text environment (NChain-v0) from OpenAI Gym. Whilst the original Rainbow paper used the Atari Learning Environment, we opted to use the Classic Control and NChain environments, as it allowed us to obtain our results promptly. In the Rainbow paper, it took their agent 10 days to fully train for a single game, whereas when using classic control, for a single run, we aimed to have our agent fully trained in much less time.

The choice of using Classic Control over ALE has some setbacks, however. The original Rainbow algorithm was developed with the Atari Learning Environment in mind, therefore we had to change some of the parameters used in the Rainbow paper, such as episode length and total frames used, in order to account for the change in environment. These changes mean that the Rainbow algorithm may produce worse results than those in the Atari Learning Environment, with respect to a random agent. However, because we aren’t changing our environment and hyper-parameters between our own agents, we can assume this to be a fair test. Another setback is that because the episode lengths of classic control environments are orders of magnitude smaller than that of the ALE (The Rainbow paper used 108K maximum frames per episode, CartPole-v0 uses 200 frames per episode), observing simply the highest reward achieved by Bootstrapped-Rainbow may not be as informative as looking at the behaviour of the algorithm, as we are attempting to observe the exploration properties of Bootstrapped-Rainbow. For this reason, we also used the NChain environment from OpenAI to test our agents.

A. Evaluation - Classic Control

We trained and evaluated our agents on 2 Classic Control environments, Cartpole-v0 and Acrobot-v1. We follow the training and evaluation procedures used in the original Rainbow paper, those of Mnih et al. (2015) and van Hasselt et al. (2016), adapting the total number of frames and evaluation periods to coincide with our smaller episode lengths due to the change of testing environment [2] [7] [17]. The average evaluation reward of each agent is calculated every 1.5K frames, by suspending training and evaluating the latest agent for 10 episodes, then taking the mean of the reward of those 10 episodes. Episodes have a maximum frame limit of 200 for Cartpole-v0 and 500 for Acrobot-v1, as detailed in the OpenAI gym Classic Control documentation.

When it came to evaluating our Bootstrapped-Rainbow algorithm, we evaluated using all heads, and recorded the averaged rewards and Q-values of each head. We then recorded the maximum reward and Q-value over all heads for that evaluation period.

B. Evaluation - NChain

NChain-v0 is an environment from Toy Text that is used to test exploration in an RL algorithm. It consists of a linear chain of states, in which the agent can move only forwards or backwards, furthermore, the agent receives a small reward for travelling backwards, and no reward for travelling forwards. That is until the agent reaches the end of the chain, at which is a large reward, however, to get there, the agent needs to traverse a large area with no rewards, thereby showcasing the exploration properties of the agent being tested. This evaluation methodology was used in the original Bootstrapped DQN paper to showcase its exploration abilities [14]. We evaluated our agent in this environment much in the same way as in our Classic Control environments. We trained and evaluated the agent at regular intervals, and, for Bootstrapped-Rainbow, evaluated over all heads and recorded the reward values for each, then recorded the maximum of these reward values for that evaluation period. We have changed some parameters, because of the change of environment. We ran this test with a total frame count of 10K frames, an episode length of 500, and evaluated each agent every 500 frames.

C. Parameter tuning

Since the Rainbow algorithm is made up of several components, there are too many combinations of parameters for an exhaustive search of the best parameter scheme. Therefore, we have left many of the parameters the same as those in the Rainbow paper, and where reasonable, have changed parameters such as minimum history to start learning.

We have increased the ϵ-greedy to 0.001 for our experiments, as without this Rainbow with no Noisy DQN would have no exploration methodology and may not have the most optimal results. Since we’ve changed it for one agent, we apply it to all of the agents for fairness.

For Bootstrapped-Rainbow, we decided on using 10 heads, as was selected in the original Bootstrapped DQN paper. This

Parameter	Value
Min history to start learning	1K Frames
Adam learning rate	0.0000625
Exploration ϵ	0.001
Noisy Nets σ_0	0.5
Target Network Period	60 Frames
Adam ϵ	1.5×10^{-4}
Prioritisation type	proportional
Prioritised exponent ω	0.5
Prioritisation importance sampling β	$0.4 \longrightarrow 1.0$
Multi-step returns n	3
Distributional atoms	51
Distributional min/max values	[-10, 10]
Number of bootstrap heads, k	10

TABLE I: Hyper-parameters used in Rainbow, Bootstrapped-Rainbow and No-Noisy-DQN Rainbow for Cartpole-v0 and Acrobot-v1

is in line with the rest of the Rainbow algorithm, as in the Rainbow paper, they started with the parameters used in each component's respective paper.

VI. Results and Analysis

In this section, we present the results of using Bootstrapped-Rainbow in Cartpole, Acrobot and NChain. We will compare Bootstrapped-Rainbow's results against that of our 2 other Agents, Rainbow, and Rainbow with no Noisy DQN.

A. Classic Control - Cartpole and Acrobot

The results for Cartpole (Fig. 1) and Acrobot (Fig 3.) show the average evaluation rewards for each agent tested over the course of 150K frames, and contain ∼100 evaluation points. At each data point, the agent was evaluated 10 times, and the mean reward value was plotted. In the case of Bootstrapped-Rainbow, every head was evaluated, as described, and the maximum value for all heads was plotted. Note that for each algorithm, 1000 random steps preceded the training process, to build up a prioritised memory buffer.

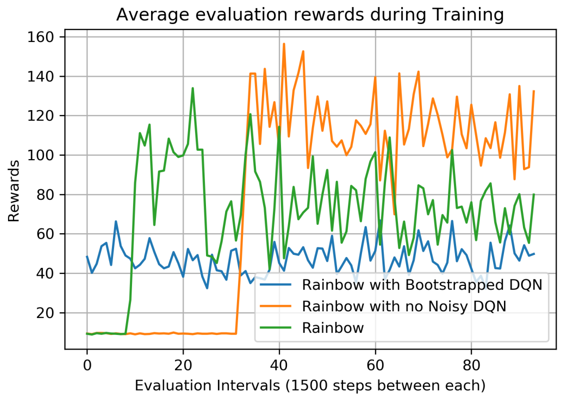

Fig. 1: The rewards obtained for Cartpole-v0 during evaluation periods whilst training

The reasoning for taking the maximum value is that each head is exploring the state space differently. Many of the heads are deeply exploring sub-optimal actions which do not lead to any reward, and so, their reward values will be lower than others when evaluated. This phenomenon is observable in Figure 2 and Figure 4, where we plot the averaged reward values of each head. A notable trend of the bootstrapped head reward values is that there is no visible positive or negative correlation in the reward obtained with more frames. Each head's reward tends to stay within a certain range, but does not go up or down in an overall trend.

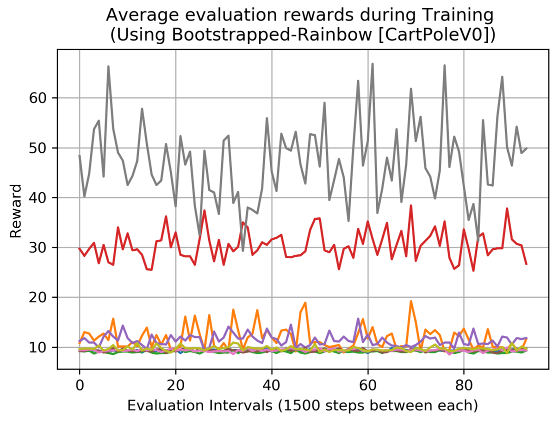

Fig. 2: The rewards obtained for Cartpole-v0 during evaluation periods whilst training for each Bootstrap head

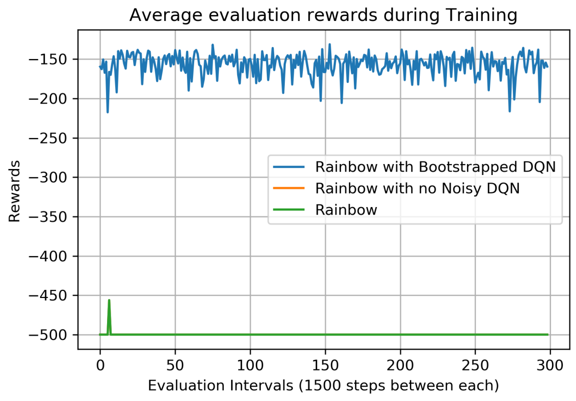

Fig. 3: The rewards obtained for Acrobot-v1 during evaluation periods whilst training

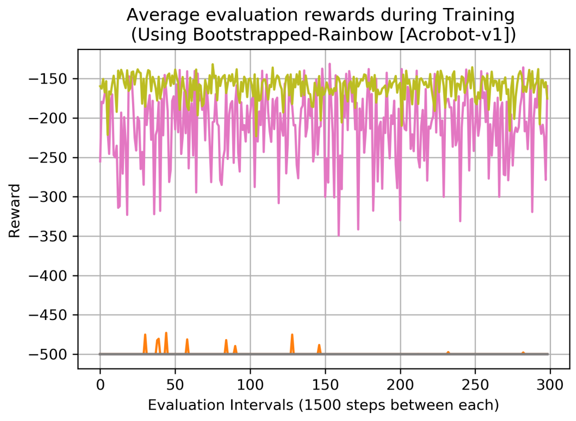

Fig. 4: The rewards obtained for Acrobot-v1 during evaluation periods whilst training for each Bootstrap head

This is likely another artefact of the Bootstrap in action, as the bootstrap explores more, it tends not to exploit the information it finds, at least with our testing hyper-parameters. Bootstrapped-Rainbow outperformed both Rainbow and Rainbow with no Noisy DQN for Acrobot, but did not for Cartpole. This is due to the change in the environment. Cartpole is a task that seems to not incentivise exploration, and favours more immediate exploitation of the data at hand. This is backed up by the fact that Rainbow with no Noisy DQN (But still with ϵ-greedy) outperformed Rainbow on this task, as Noisy DQN is currently only one of two exploration techniques implemented in Rainbow, and it performed better on this task without it. Therefore, it is reasonable to assume that this is the reason for Bootstrapped-Rainbow's lower score in this environment. Acrobot, on the other hand, seems to be a task where deep exploration of states and actions is incentivised, as Bootstrapped-Rainbow performs well on this task, outperforming Rainbow and Rainbow with no Noisy DQN by a wide margin. This seems to makes sense, as in the Cartpole environment, the agent must react to the pole dropping of its own accord, whereas in Acrobot, the agent must provoke the environment into a different state. Also, in Acrobot, a negative reward is given until 500 frames have passed or the agent gets the end of the 2nd stick above the threshold line. This allows the agent that can explore options more deeply in the limited episode length to discover a better reward. Without this, the reward for an episode is -500.

B. NChain

To show that Bootstrapped-Rainbow can explore deeper than Rainbow or Rainbow with no Noisy DQN, the best environment is one in which we can have a sparse, large reward that is not found until the agent has passed many states which do not provide any reward. NChain is a good candidate for this, and was used in the original Bootstrapped DQN paper to demonstrate this. We can see clearly from Figure 5 that Bootstrapped-Rainbow is exploring deeper than any of the other two algorithms, because in order for it to obtain such a high reward, it would have to make it to the end of the chain in a certain number of frames. Rainbow and No Noisy DQN Rainbow have fallen victim to their lack of deep exploration, as they decide to go for the action which will provide them with small, quasi-immediate reward, instead of searching deeper for a larger, more substantial reward. Also note that the majority of the bootstrap heads produce a reward that is 15,000 or higher for a 500 episode run, implying that all heads are searching deeply for a more rewarding strategy, and is not only one or two heads searching, as it may appear in Figures 2 and 4 from the Classic Control experiments.

C. Exploration vs Exploitation, Different Domains

The recurring theme of these results and comparisons is that of exploration vs exploitation. Exploration being the seeking out of information about the state space, such as rewards, actions taken to get there, and exploitation being the use of information we have on hand to maximise our reward values. Bootstrapped-Rainbow seems to over-explore in our Cartpole-v0 experiment, when compared to Rainbow and No Noisy DQN Rainbow, and does not use the information it has already to pick a policy which it can then exploit to produce better reward values. We know that it is searching for a higher reward, as shown in NChain-v0, and in our Acrobot-v0 experiment, it found a policy which produced better results than our other two algorithms. This shows that the use of Bootstrapped-Rainbow is suited more to domains with sparse rewards than to those which require a large amount of exploitation, although there is still an issue which requires more investigation, and that is that no matter if Bootstrapped-Rainbow produces good results for a particular environment, there seems to be no overall trend upwards or downwards, but the values seem to sit around the same level.

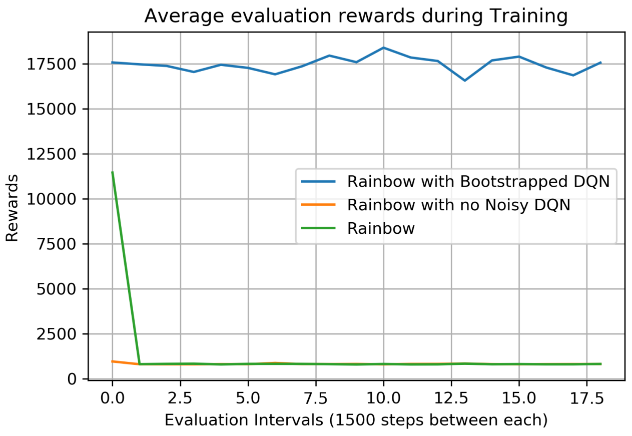

Fig. 5: The rewards obtained for NChain-v0 during evaluation periods whilst training

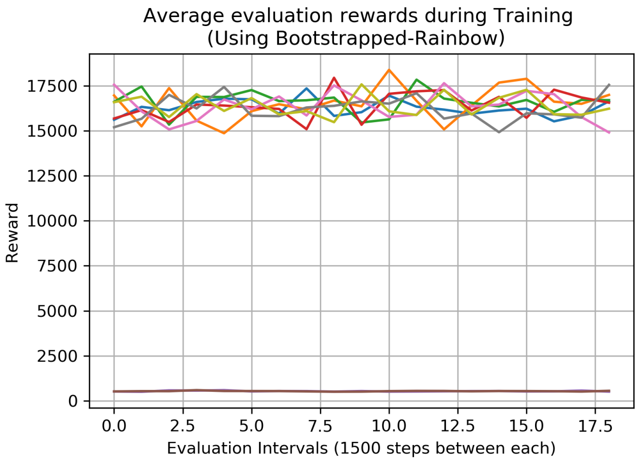

Fig. 6: The rewards obtained for each head in Bootstrapped-Rainbow using NChain-v0 during evaluation periods whilst training

VII. Conclusion

In this paper we have shown that Bootstrapped-DQN can be a very useful extension to the Rainbow reinforcement learning algorithm, in particular domains where exploration is encouraged, such as Acrobot, or NChain. Bootstrapped-Rainbow goes into deeper exploration of the state space than

Rainbow, but at the cost of lack of exploitation, as can be seen with the flat overall gradients in Figures 1, 3 and 5. It is more suited to environments which have sparse rewards, when compared to Rainbow, or Rainbow with No Noisy DQN, which can be seen with Acrobot, and NChain. Further studies could look into as to why this flat gradient occurs, and aim to tackle that issue.

This study did not, however, investigate any of the Atari Learning Environment, which Rainbow was developed with in mind, and there are some games in the ALE which do have sparse rewards, one example would be Montezuma's Revenge. However, as a general-purpose algorithm, I do not believe Bootstrapped-Rainbow is an ideal algorithm. It is however, very useful in those situations where it is needed, for domains which encourage exploration, and have sparse rewards. Perhaps if there was a method of quantifying how rewarding a testing environment was for exploring, then one could look at implementing Bootstrapped-Rainbow into a general-purpose Rainbow algorithm, and use it once a certain "Exploration viability" threshold had been reached. However, for the moment, Bootstrapped-Rainbow is not a general purpose RL algorithm, and should instead be used for those cases where it is deemed viable.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcment learning," *DeepMind Technologies, pp*, pp. 1–9, 2013.

[2] M. Hessel, T. S. G. O. W. D. D. H. B. P. M. A. J. Modayil, H. v. Hasselt, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *DeepMind*, pp. 1–14, 2017.

[3] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[4] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[5] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[6] C. J. Watkins and P. Dyan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[7] D. S. A. A. R. J. V. M. G. B. A. G. M. R. A. K. F. G. O. S. P. C. B. A. S. I. A. H. K. V. Mnih, K. Kavukcuoglu and E. al., "Human level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[8] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," *CUED/F-INFENG/TR*, vol. 166, 1991.

[9] D. Isele and A. Cosgun, "Selective experience replay for lifelong learning," in *The Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 3302–3309.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Ward-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Département d'informatique et de recherche opérationnelle, Université de Montréal, pp*, pp. 1–9, 2014.

[11] V. Huang, T. Ley, M. Vlachou-Konchylaki, and M. Hu, "Enhanced experience replay generation for efficient reinforcement learning," *Ericsson AB, pp*, pp. 1–7, 2017.

[12] C. Lyle, T. Doan, and B. Mazoure, "Gan q-learning," *McGill University, pp*, pp. 1–10, 2018.

[13] D. Pfau and O. Vinyals, "Connecting generative adversarial networks and actor-critic methods," *Google DeepMind, pp*, pp. 1–10, 2017.

[14] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep exploration via bootstrapped dqn," *Stanford University, Google DeepMind, pp*, pp. 1–18, 2016.

[15] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," pp. 1–19, 2017.

[16] B. P. e. a. Meire Fortunato, Mohammad Gheshlaghi Azar, "Noisy networks for exploration," pp. 1–19, 2018.

[17] A. van Hasselt, H.; Guez and D. Silver, "Deep reinforcement learning with double q-learning," *Proc. of AAAI*, pp. 2094–2100, 2016.