Deep Exploration via Bootstrapped DQN

Ian Osband[1,2] IOSBAND@GOOGLE.COM
Charles Blundell[1] CBLUNDELL@GOOGLE.COM
Alexander Pritzel[1] APRITZEL@GOOGLE.COM
Benjamin Van Roy[2] BVR@STANFORD.EDU
[1]Google DeepMind, [2]Stanford University

Abstract

Efficient exploration in complex environments remains a major challenge for reinforcement learning. We propose *bootstrapped DQN*, a simple algorithm that explores in a computationally and statistically efficient manner through use of randomized value functions. Unlike dithering strategies such as ϵ-greedy exploration, bootstrapped DQN carries out temporally-extended (or deep) exploration; this can lead to exponentially faster learning. We demonstrate these benefits in complex stochastic MDPs and in the large-scale Arcade Learning Environment. Bootstrapped DQN substantially improves learning times and performance across most Atari games.

1. Introduction

We study the reinforcement learning (RL) problem where an agent interacts with an unknown environment. The agent takes a sequence of actions and learns from observations and rewards. The goal is to maximize cumulative rewards. Unlike standard planning problems, an RL agent does not begin with perfect knowledge of the environment, but learns through experience. This leads to a fundamental trade-off of exploration versus exploitation; the agent may improve its future rewards by exploring poorly understood states and actions, but this may require sacrificing immediate rewards.

To learn efficiently an agent should explore only when there are valuable learning opportunities. Common heuristics such as ϵ-greedy and Boltzmann exploration leave enormous room for improvement. Since such algorithms explore at random times to balance effort between exploration and exploitation, we call these dithering strategies. Further, as the consequences of an action may extend over multiple timesteps, the agent should reason about the informational value of possible observation sequences. Without this sort of temporally extended (deep) exploration, learning times can worsen by an exponential factor.

The theoretical RL literature offers a variety of provably-efficient approaches to deep exploration (e.g., Jaksch et al. (2010); Guez et al. (2012)). However, most of these are designed for Markov decision processes (MDPs) with small finite state spaces, while others require solving computationally intractable planning tasks. These algorithms are not practical in complex environments where an agent must generalize to operate effectively.

For this reason, large-scale applications of RL have relied upon dithering exploration strategies which are statistically inefficient. These methods have lead to several high-profile applications with groundbreaking results (Tesauro, 1995; Mnih, 2015). Nevertheless, we might hope to improve upon these exploration heuristics. We review related literature in more detail in Section 4.

In this paper, we consider an alternative approach to exploration based on randomized value functions. Previous work demonstrated that randomized value functions can efficiently explore in tandem with linearly-parameterized value function generalization (Osband et al., 2014). We present a natural and principled extension of this approach that enables use of complex non-linear generalization methods such as deep neural networks.

Our paper begins with a review of how to represent uncertainty with neural networks. We show that the bootstrap with random initialization can produce reasonable uncertainty estimates for neural networks at low computational cost. In Section 3 we present our algorithm, *bootstrapped deep Q networks (DQN)*, which leverages these uncertainty estimates for efficient (and deep) exploration. We review related work in Section 4. Section 5 presents a series of didactic experiments that are designed to highlight the importance of deep exploration and that are intractable for any shallow exploration strategy. Bootstrapped DQN succeeds in tasks that require deep exploration extending over 100 time periods and even in difficult stochastic MDPs. Finally we demonstrate that these benefits can extend to large scale problems that are not designed to highlight deep exploration. Section 6 presents our results from application to

Atari 2600. Bootstrapped DQN substantially reduces learning times and improves performance across most games. This algorithm is computationally efficient and amenable to parallelization; on a single machine our implementation runs roughly 20% slower than DQN.

2. Uncertainty for neural networks

Deep neural networks represent the state of the art in many supervised learning domains (Krizhevsky et al., 2012). This is largely due to their flexibility, scalability and inductive bias, which allows them to learn effective feature representations. The exploration method we study in this paper is designed to be statistically and computationally efficient when used in conjunction with neural network representations of the value function.

To explore efficiently, it is important to quantify uncertainty in value estimates so that the agent can judge potential benefits of exploratory actions. The neural network literature presents a sizable body of work on uncertainty quantification founded on parametric Bayesian inference. These include variational Bayes (Graves, 2011; Blundell et al., 2015), assumed density filtering (Hernández-Lobato & Adams, 2015), dropout-based variational inference (Gal & Ghahramani, 2015; Kingma et al., 2015), and stochastic gradient Langevin dynamics (Teh et al., 2015). Instead of appealing to parametric assumptions about the uncertainty present in RL problems, we use the non-parametric bootstrap (Efron, 1982) to obtain a distribution over functions represented by the neural network.

There are several variants of the bootstrap (Efron & Tibshirani, 1994), each of which relies upon data-based simulation. The key idea is to approximate a population distribution by a sample distribution (Efron & Tibshirani, 1994). In its most common form, the bootstrap takes as input a data set D and an estimator ψ. To generate a sample from the bootstrapped distribution, a data set $\tilde{D}$ of cardinality equal to that of D is sampled uniformly with replacement from D. The bootstrap sample estimate is then taken to be $\psi(\tilde{D})$.

The bootstrap is widely hailed as a great advance of 20th century applied statistics and comes with theoretical guarantees (Bickel & Freedman, 1981). Although the bootstrap was introduced as a frequentist method, several variants have natural interpretations under Bayesian and empirical Bayes analysis (Rubin et al., 1981). In some settings, when the original data set is augmented with appropriate synthetic samples, bootstrap samples are distributed according to the posterior distribution for an appropriate prior (Osband & Van Roy, 2015). The bootstrap is highly parallelizable and, as such, amenable to distributed computation. The approach can even scale to massive data with sub-linear computational cost (Kleiner et al., 2014).

In Figure 1 we present an efficient and scalable method for generating bootstrap samples from a large and deep neural network. The network consists of a shared architecture with K bootstrapped "heads" branching off independently. Each head is trained only on its bootstrapped sub-sample of the data, which can be generated online (Owen et al., 2012). Thus each head represents $\psi(\tilde{D})$ in the classical view of bootstrap. The shared network learns a joint feature representation, this can provide significant computational advantages at the cost of lower diversity between heads. This type of bootstrap can even be implemented efficiently by a single forward/backward pass of backpropagation; it can be thought of as a data-dependent dropout, where the dropout mask for each head is fixed for each data point (Srivastava et al., 2014).

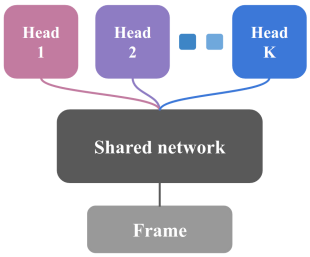
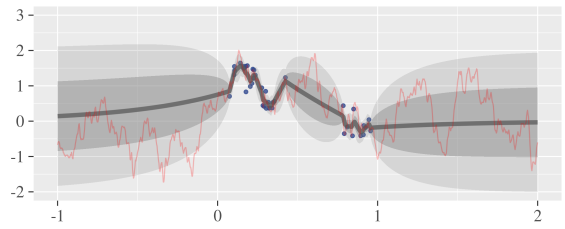
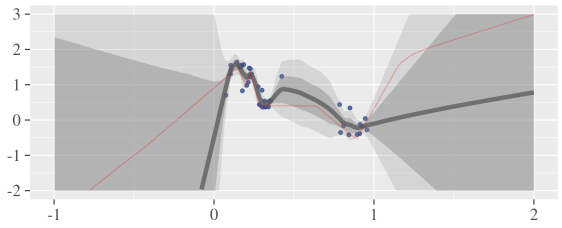


Figure 1: An efficient architecture for K bootstrap samples

Figure 2 presents an example of uncertainty estimates from bootstrapped neural networks on a regression task with noisy data. We trained a fully-connected 2-layer neural networks with 50 rectified linear units (ReLU) in each layer on 50 bootstrapped samples from the data. As is standard practice, we initialize these networks with random parameter values, this induces an important initial diversity in the models. We were unable to generate effective uncertainty estimates for this problem using the dropout approach in prior literature (Gal & Ghahramani, 2015). Further details are provided in Appendix A.



(a) Gaussian process posterior



(b) Bootstrapped neural networks

Figure 2: Grey regions depict the mean estimate $\pm\{1, 2\}$ standard deviations. A single sample is shown in red.

3. Bootstrapped DQN

For a policy π we define the value of an action a in state s

$$Q^\pi(s,a) := \mathbb{E}_{s,a,\pi}\left[\sum_{t=1}^{\infty}\gamma^t r_t\right],$$

where $\gamma \in (0,1)$ is a discount factor that balances immediate versus future rewards r_t. This expectation indicates that the initial state is s, the initial action is a, and thereafter actions are selected by the policy π. The optimal value is $Q^*(s,a) := \max_\pi Q^\pi(s,a)$. To scale to large problems, we learn a parameterized estimate of the Q-value function $Q(s,a;\theta)$ rather than a tabular encoding. We use a neural network to estimate this value.

The Q-learning update after taking action a_t in state s_t and observing reward r_t and transitioning to s_{t+1} is given by

$$\theta_{t+1} \leftarrow \theta_t + \alpha(y_t^Q - Q(s_t,a_t;\theta_t))\nabla_\theta Q(s_t,a_t;\theta_t) \quad (1)$$

where α is the scalar learning rate and y_t^Q is the target value $r_t + \gamma \max_a Q(s_{t+1},a;\theta^-)$. θ^- are target network parameters fixed $\theta^- = \theta_t$.

Several important modifications to the Q-learning update improve stability for DQN (Mnih, 2015). First the algorithm learns from sampled transitions from an experience buffer, rather than learning fully online. Second the algorithm uses a target network with parameters θ^- that are copied from the learning network $\theta^- \leftarrow \theta_t$ only every τ time steps and then kept fixed in between updates. Double DQN (DDQN) modifies the target y_t^Q:

$$y_t^Q \leftarrow r_t + \gamma \max_a Q\big(s_{t+1}, \arg\max_a Q(s_{t+1},a;\theta_t);\theta^-\big).$$

DDQN can demonstrate improved performance and stability (Van Hasselt et al., 2015). In this paper we use the DDQN update for all DQN variants unless explicitly stated.

Our algorithm, *bootstrapped DQN* in Algorithm 1, modifies DQN to produce *distribution* over Q-values via the bootstrap. At the start of each episode, bootstrapped DQN samples a single Q-value function from its approximate posterior. The agent then follows the policy which is optimal for that *sample* for the duration of the episode. This is a natural extension of the Thompson sampling heuristic to RL that allows for temporally extended (or deep) exploration (Strens, 2000; Osband et al., 2013).

In order to implement this algorithm efficiently online we build up $K \in \mathbb{N}$ bootstrapped estimates of the Q-value function in parallel as in Figure 1. Importantly, each one of these value function function heads $Q_k(s,a;\theta)$ is trained against its own target network $Q_k(s,a;\theta^-)$. This means that each $Q_1,..,Q_K$ provide a temporally extended (and consistent) estimate of the value uncertainty via TD estimates. We approximate a bootstrap sample by selecting $k \in \{1,..,K\}$ uniformly at random and following Q_k for the duration of that episode. Bootstrapped DQN exhibits deep exploration unlike the naive application of Thompson sampling to RL which resample every timestep[1].

Algorithm 1 Bootstrapped DQN

Input: neural network $(Q_k)_{k=1}^K$, sampling distribution P
for each episode **do**
  Update network parameters via minibatches
  Sample $k \sim$ Uniform$\{1,...,K\}$
  while not end of episode **do**
    Choose $a_t \in \text{argmax}_a Q_k(s_t,a)$
    Receive state s_{t+1} and reward r_t from environment
    Sample bootstrap mask $m_t^k \sim P$ for all k
    Add (a_t,r_t,s_{t+1},m_t) to replay buffer
  end while
end for

4. Related work

The observation that temporally extended exploration is necessary for efficient reinforcement learning is not new. In fact, for any prior distribution over MDPs, the optimal exploration strategy is available through dynamic programming in the Bayesian belief state space. However, the exact solution is intractable even for very simple systems (Burnetas & Katehakis, 1997). RL algorithms provide tractable approximations to this problem.

Most common RL methods focus on generalization and planning, but address exploration via dithering strategies. Many successful applications employ only ϵ-greedy exploration (Tesauro, 1995; Mnih, 2015). However, such exploration strategies can be highly inefficient, as we demonstrate in Section 5.

Policy gradient methods offer another approach to RL and induce random exploration since they search over the space of stochastic policies. Although there are several successful applications of policy gradient methods (e.g., Levine et al. (2015)), the form of exploration induced by such algorithms can also be highly inefficient (Kakade, 2003).

Other approaches aim to approximate Bayes-optimal exploration though tree-based search (Wang et al., 2005; Guez et al., 2014). These approaches are well-founded and, with unlimited computation will converge to the optimal policy. However, practical implementations with finite computation may fail dramatically (Munos, 2014).

Many exploration strategies are guided by the principle of "optimism in the face of uncertainty" (OFU). These algorithms add an exploration bonus to values of state-action pairs that may lead to useful learning and select actions to maximize these adjusted values. This approach was first

[1] We call bootstrapped DQN which resamples a head every timestep Thompson DQN. This is similar to to the "Thompson sampling" algorithm of (Gal & Ghahramani, 2015) but uses the bootstrap in place of dropout as an approximate posterior.

proposed for finite-armed bandits (Lai & Robbins, 1985), but the principle has been extended successfully across bandits with generalization (Rusmevichientong & Tsitsiklis, 2010) and tabular RL (Jaksch et al., 2010). Except for particular deterministic contexts (Wen & Van Roy, 2013), OFU methods that lead to efficient RL in complex domains have been computationally intractable. The work of (Stadie et al., 2015) aims to add an effective bonus through a variation of DQN. The resulting algorithm relies on a large number of hand-tuned parameters and is only suitable for application to deterministic problems. We compare our results on Atari to theirs in Appendix C.

Perhaps the oldest heuristic for balancing exploration with exploitation is given by Thompson sampling (Thompson, 1933). This bandit algorithm takes a single sample from the posterior at every time step and chooses the action which is optimal for that time step. Thompson sampling offers good performance in bandits and several state of the art guarantees (Russo & Van Roy, 2014).

To apply the Thompson sampling principle to RL, an agent should sample a value function from its posterior. The agent must also commit to this sample for several time steps in order to achieve deep exploration (Strens, 2000; Guez et al., 2012). The algorithm PSRL does exactly this, with state of the art guarantees (Osband et al., 2013; Osband & Van Roy, 2014b;a; Abbasi-Yadkori & Szepesvári, 2015; Gopalan & Mannor, 2015). However, this algorithm still requires solving a single known MDP, which will usually be intractable for large systems.

Our new algorithm, bootstrapped DQN, approximates this approach to exploration via randomized value functions sampled from an approximate posterior. Recently, authors have proposed the RLSVI algorithm which accomplishes this for linearly parameterized value functions. Surprisingly, this algorithm recovers state of the art guarantees in the setting with tabular basis functions, but its performance is crucially dependent upon a suitable linear representation of the value function (Osband et al., 2014). We extend these ideas to produce an algorithm that can simultaneously perform generalization and exploration with a flexible nonlinear value function representation. Our method is simple, general and compatible with almost all advances in deep RL at low computational cost and with few tuning parameters.

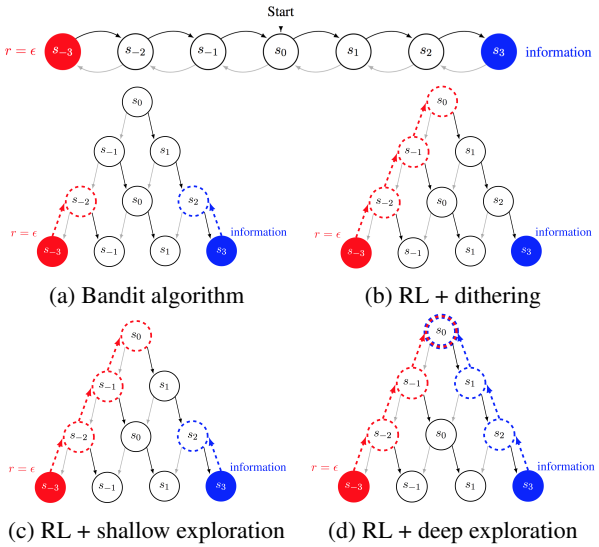
5. Deep Exploration

Uncertainty estimates allow an agent to direct its exploration at potentially informative states and actions. In bandits, this choice of directed exploration rather than dithering generally categorizes efficient algorithms. The story in RL is not as simple, directed exploration is not enough to guarantee efficiency; the exploration must also be deep.

Deep exploration means exploration which is directed over multiple time steps; it can also be called "planning to learn" or "far-sighted" exploration. Unlike bandit problems, which balance actions which are immediately rewarding or immediately informative, RL settings require planning over several time steps (Kakade, 2003). For exploitation, this means that an efficient agent must consider the future rewards over several time steps and not simply the myopic rewards. In exactly the same way, efficient exploration may require taking actions which are neither immediately rewarding, nor immediately informative.

To illustrate this distinction, consider a simple deterministic chain $\{s_{-3}, .., s_{+3}\}$ with three step horizon starting from state s_0. This MDP is known to the agent a priori, with deterministic actions "left" and "right". All states have zero reward, except for the leftmost state s_{-3} which has known reward $\epsilon > 0$ and the rightmost state s_3 which is unknown. In order to reach either a rewarding state or an informative state within three steps from s_0 the agent must plan a consistent strategy over several time steps.

Figure 3 depicts the planning and look ahead trees for several algorithmic approaches in this example MDP. The action "left" is gray, the action "right" is black. Rewarding states are depicted as red, informative states as blue. Dashed lines indicate that the agent can plan ahead for either rewards or information. Unlike bandit algorithms, an RL agent can plan to exploit future rewards. Only an RL agent with deep exploration can plan to learn.



(a) Bandit algorithm (b) RL + dithering

(c) RL + shallow exploration (d) RL + deep exploration

Figure 3: Planning, learning and exploration in RL.

5.1. Testing for deep exploration

We now present a series of didactic computational experiments designed to highlight the need for deep exploration in RL. The agents will be placed in an environment made

up of a long chain of states right next to a small reward. All other states have zero reward except at the far end of the chain where they can find a state with much higher reward.

The experiments in this section are toy problems intended to be expository rather than entirely realistic. However, they clearly test whether an agent can efficiently balance the potential benefits of delayed information from deep exploration. Balancing a well known and mildly successful strategy versus an unknown, but potentially more rewarding, approach can emerge in many practical applications.

5.2. Learning from pixels

The environments in this section may be concisely described by a finite tabular MDP, similar to *RiverSwim* (Strehl & Littman, 2005). However, we will require our algorithm to interact with the MDP only through raw pixel features. For a chain of length N we will define features of the state $\phi : \{1, .., N\} \to \{0, 1\}^N$. We consider two feature mappings, "one-hot" $\phi_{\text{oh}}(s_t) := (\mathbb{1}\{x = s_t\})_{x=1,..,N}$ and "thermometer" $\phi_{\text{therm}}(s_t) := (\mathbb{1}\{x \leq s_t\})_{x=1,..,N}$.

We find that bootstrapped DQN is able to explore efficiently with either set of features. We focus upon the thermometer encoding, since this better captures generalization between states. Full details for these experiments are given in Appendix B.

5.3. Scaling deeper

We consider a family of deterministic chains of length $N > 3$ as in Figure 4. Each episode of interaction lasts $N + 9$ steps after which point the agent resets to the initial state s_2. We choose this so that the optimal policy is to move right at every step and receive a return of 10 in each episode. However, any shallow exploration strategy will take $\Omega(2^N)$ episodes to learn the optimal policy (Osband et al., 2014).

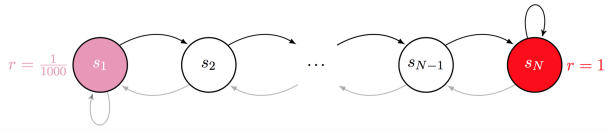


Figure 4: Scalable environments that requires deep exploration.

We say that the algorithm has successfully learned the optimal policy when it has successfully completed one hundred episodes with optimal reward of 10. For each chain length, we ran each learning algorithm for 2000 episodes across three seeds. We plot the median time to learn in Figure 5, together with a conservative lower bound of $99 + 2^{N-11}$ on the expected time to learn for any shallow exploration strategy. Only bootstrapped DQN demonstrates a graceful scaling to long chains which require deep exploration.

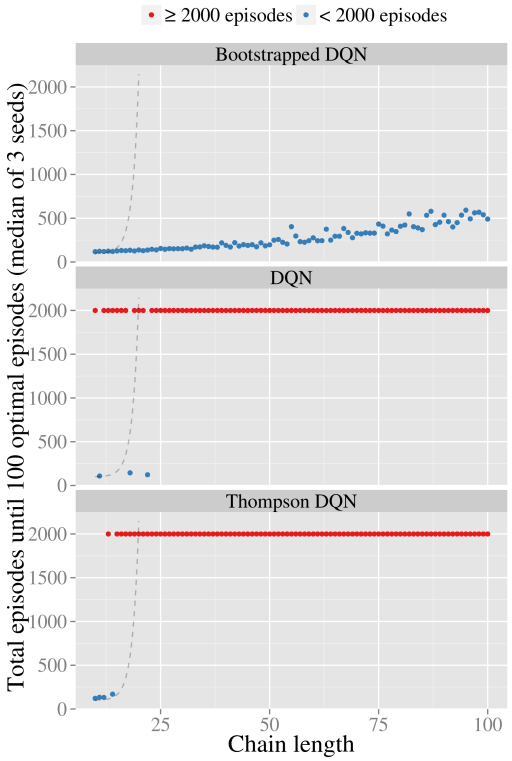


Figure 5: Bootstrapped DQN demonstrates deep exploration.

5.4. A difficult stochastic MDP

Figure 5 shows that bootstrapped DQN can implement effective (and deep) exploration where similar deep RL architectures fail. However, since the underlying system is a small and finite MDP there may be several other simpler strategies which would also solve this problem. We will now consider a difficult variant of this chain system with significant stochastic noise in transitions as depicted in Figure 6. Action "left" deterministically moves the agent left, but action "right" is only successful 50% of the time and otherwise also moves left. The agent interacts with the MDP in episodes of length 15 and begins each episode at s_1. Once again the optimal policy is to head right.

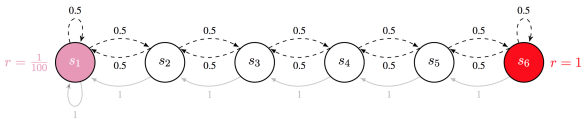


Figure 6: A stochastic MDP that requires deep exploration.

Bootstrapped DQN is unique amongst scalable approaches to efficient exploration with deep RL in stochastic domains. For benchmark performance we implement three algorithms which, unlike bootstrapped DQN, will receive the true tabular representation for the MDP. These algorithms are based on three state of the art approaches to exploration via dithering (ϵ-greedy), optimism (UCRL2; Jaksch et al., 2010) and posterior sampling (PSRL; Osband et al., 2013). We discuss the choice of these benchmarks in Appendix B.

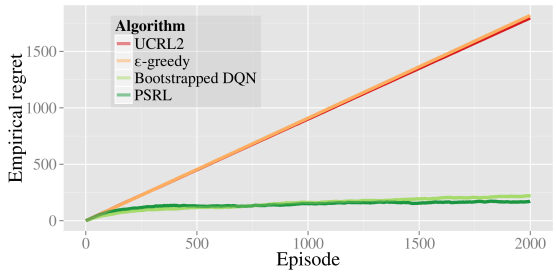


Figure 7: Bootstrapped DQN matches efficient tabular RL.

In Figure 7 we present the empirical regret of each algorithm averaged over 10 seeds over the first two thousand episodes. The empirical regret is the cumulative difference between the expected rewards of the optimal policy and the realized rewards of each algorithm. We find that bootstrapped DQN achieves similar performance to state of the art efficient exploration schemes such as PSRL even without prior knowledge of the tabular MDP structure and in noisy environments.

Most telling is how much better bootstrapped DQN does than the state of the art optimistic algorithm UCRL2. Although Figure 7 seems to suggest UCRL2 incurs linear regret, actually it follows its bounds $\tilde{O}(S\sqrt{AT})$ (Jaksch et al., 2010) where S is the number of states and A is the number of actions. We demonstrate this through simulation in Appendix B, together with extended discussion.

6. Arcade Learning Environment

We now evaluate our algorithm across 49 Atari games on the Arcade Learning Environment (Bellemare et al., 2012). Importantly, and unlike the experiments in Section 5, these domains are not specifically designed to showcase our algorithm. In fact, many Atari games are structured so that small rewards always indicate part of an optimal policy. This may be crucial for the strong performance observed by dithering strategies[2]. We find that exploration via bootstrapped DQN produces significant gains versus ϵ-greedy in this setting. Bootstrapped DQN reaches peak performance roughly similar to DQN. However, our improved exploration mean we reach human performance on average 30% faster across all games. This translates to significantly improved cumulative rewards through learning.

We follow the setup of (Van Hasselt et al., 2015) for our network architecture and benchmark our performance against their algorithm. Our network structure is identical to the convolutional structure of DQN (Mnih, 2015) except we split 10 separate bootstrap heads after the convolutional layer as per Figure 1. Recently, several authors have provided architectural and algorithmic improvements to DDQN (Wang et al., 2015; Schaul et al., 2015). We do not compare our results to these since their advances are orthogonal to our concern and could easily be incorporated to our bootstrapped DQN design. Full details of our experimental set up are available in Appendix C.

[2]By contrast, imagine the agent received a small reward for dying; dithering strategies would be hopeless, just like Section 5.

6.1. Implementing bootstrapped DQN at scale

We now examine how to generate online bootstrap samples for DQN in a computationally efficient manner. We use a shared convolutional network architecture with K bootstrap heads as per Figure 1. This leaves three key questions: how many heads do we need, how should we pass gradients to the shared network and how should we bootstrap data online? Each of these questions involves balancing computational and statistical considerations.

We found that even a small number of bootstrap heads was sufficient to incentivize efficient exploration. Figure 8 presents the cumulative reward of bootstrapped DQN on the game Breakout, for different number of heads K. More heads leads to faster learning, but even a small number of heads captures most of the benefits of bootstrapped DQN, we choose $K{=}10$. Due to the shared convolutional network, we find this overall network retains similar computate speed to DQN.

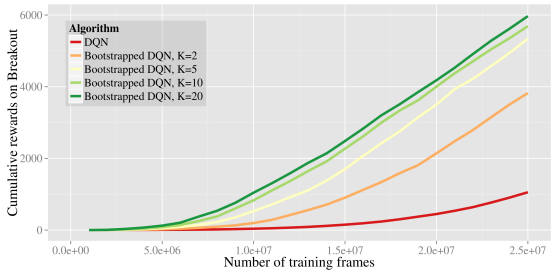


Figure 8: Decreasing marginal benefit to extra bootstrap heads.

This shared network architecture allows us to train this combined network via backpropagation. One question is whether to normalize gradients flowing from the network heads. Feeding K network heads to the shared convolutional network effectively increases the learning rate for this portion of the network. In some games, this leads to premature and sub-optimal convergence. We found the best final scores by normalizing the gradients by $1/K$, but this also leads to slower early learning. Since much of the literature focusses best policy after 200m frames, rather than cumulative rewards during learning, we choose $1/10$ rescaling for our shared gradients[3].

Finally we have the question of how to implement an online bootstrap for the K network heads. In order to avoid a significant increase in memory footprint we use an independent Bernoulli mask $w_1, .., w_K \sim \text{Ber}(p)$ for each head

[3]For more details on gradient rescaling see Appendix C.

in each episode[4]. These flags are stored in the memory replay buffer and identify which heads are trained on which data. A smaller p means that fewer data is shared, which will maintain a higher diversity of bootstrap samples.

However, when trained using a shared minibatch the algorithm will also require an effective $1/p$ more iterations. This is undesirable since DQN is already computationally taxing. Surprisingly, we found the algorithm performed similarly irrespective of p and all outperformed DQN, as shown in Figure 9. This is strange, but we present a more detailed investigation of this phenomenon in Appendix C. In light of this empirical observation for Atari, we chose $p = 1$ to save on minibatch passes. As a result bootstrapped DQN runs at similar computational speed to vanilla DQN on identical hardware[5].

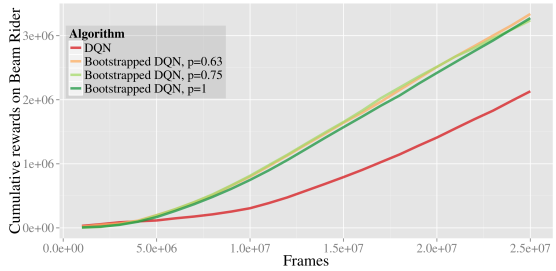
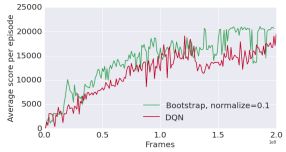


Figure 9: We found similar performance across a range of p.

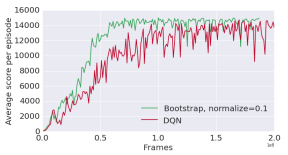
6.2. Efficient exploration in Atari

We find that Bootstrapped DQN drives efficient exploration in several Atari games. This means that, for the same amount of game experience, bootstrapped DQN generally outperforms DQN with ϵ-greedy exploration. Figure 10 demonstrates this effect for a diverse selection of games.

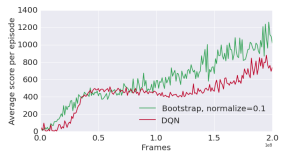
(a) Breakout



(b) Hero



(c) Qbert



(d) James Bond

Figure 10: Bootstrapped DQN drives more efficient exploration.

To summarize this improvement in learning time we consider the number of frames required to reach human performance. We say algorithm A has a human speedup of x relative to algorithm B if it reaches human performance in $1/x$ as many frames as DQN. We find that, for the majority of games where DQN reaches human performance, Bootstrapped DQN reaches human performance significantly faster. We present these results in Figure 11.

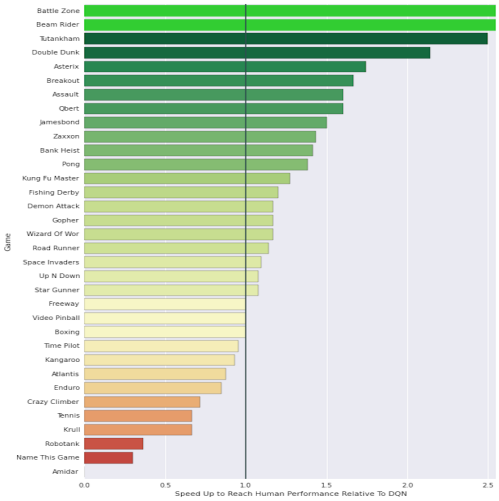


Figure 11: Bootstrapped DQN at human level faster than DQN.

6.3. Understanding bootstrapped DQN

We will now present some insight into how bootstrapped DQN uses deep exploration to improve upon DQN. First, we will examine the exploration policies for the game Hero. In this game the agent controls a rescue mission which travels room to room, blows up obstacles and rescues hostages. In Figure 12 we show a screenshot for nine different heads of bootstrapped DQN run from the initial state for 3,200 steps. Although each individual head has learned a good representation for the value function that leads to a high-scoring policy, the policies they find are quite distinct.

In fact, this screenshot shows the nine different heads in five different game rooms. This shows that even after more than 100m training frames with the data sharing $p = 1$, we still maintain significant diversity. By contrast, ϵ-greedy strategies are almost indistinguishable for small values of ϵ and totally ineffectual for larger values. Our heads explore a diverse range of policies, but still manage to each perform well individually. We believe that this deep exploration is key to the improved learning visible in Figure 10b, since diverse experiences allow for better generalization. We present videos to highlight this behavior at https://youtu.be/Zm2KoT82O_M.

Disregarding exploration, bootstrapped DQN may be beneficial as a purely exploitative policy. We can combine all the heads into a single ensemble policy, for example by choosing the action with the most votes across heads. This approach might have several benefits. First, we find that the ensemble policy can often outperform any individual policy. Second, the distribution of votes across heads to give a measure of the uncertainty in the optimal policy.

[4] $p = 0.5$ is double-or-nothing bootstrap (Owen et al., 2012).

[5] Our implementation K=10, p=1 ran with less than a 20% increase on wall-time versus DQN for the same amount of frames.

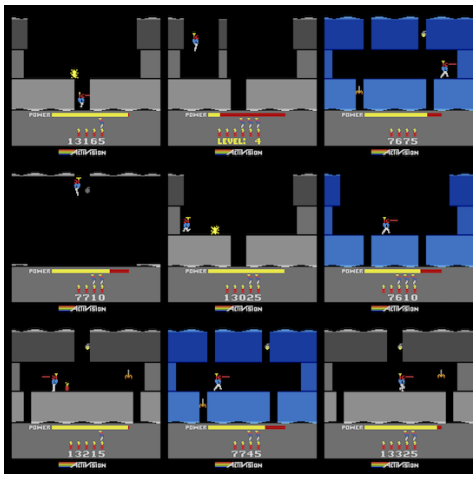
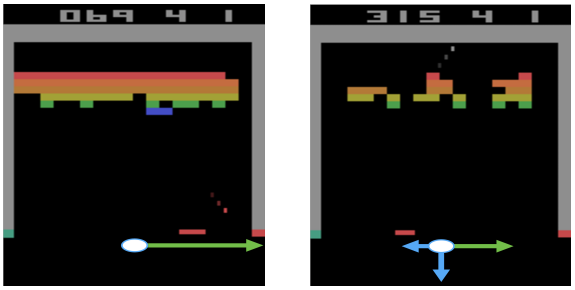


Figure 12: Diverse exploration policies in Hero.

In Figure 13 we present two screenshots of bootstrapped DQN playing Breakout according to an ensemble policy. In this game the agent can choose to move left, move right or stay. Beneath each screen we depict the number of heads that pick each action by the length of each arrow. We draw the action chosen by the ensemble policy in green.



(a) All heads vote right. (b) Heads disagree on policy.

Figure 13: Visualizing uncertainty in an ensemble policy.

When the ball is approaching the bottom of the screen and all ten heads recognize the optimal policy is to move right to catch it (Figure 13a). However when the ball is far from the paddle the we can see the ensemble policy is uncertain on the best action (Figure 13b). We present videos of this effect at `https://youtu.be/0jvEcC5JvGY`. Unlike vanilla DQN, bootstrapped DQN can know what it doesn't know. In an application where executing a poorly-understood action is dangerous this could be crucial.

6.4. Overall performance

We have seen that bootstrapped DQN is able to learn much faster than DQN. In Figure 14 we see that best performance reached by both algorithms is similar across most games. However, the benefits of efficient exploration mean that bootstrapped DQN greatly outperforms DQN when measured in terms of *cumulative* rewards through learning. We present these results in Figure 15. In Appendix C we present full results for our algorithm across all 49 games. Bootstrapped DQN significantly outperforms several recent heuristic approaches for improved exploration on Atari (Stadie et al., 2015), particularly in terms of cumulative rewards.

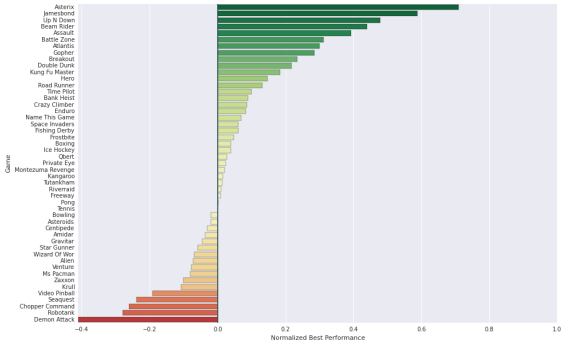


Figure 14: Bootstrapped DQN attains similar best policy.

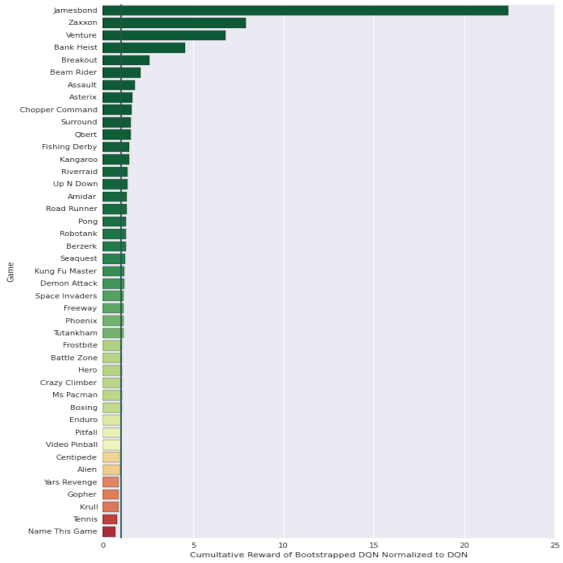


Figure 15: Bootstrapped DQN improves cumulative rewards.

7. Closing remarks

In this paper we present bootstrapped DQN as an algorithm for efficient reinforcement learning in complex environments. We demonstrate that the bootstrap can produce useful uncertainty estimates for deep neural networks. Bootstrapped DQN can leverage these uncertainty estimates for deep exploration even in difficult stochastic systems; it also produces several state of the art results in Atari 2600.

Bootstrapped DQN is computationally tractable and also naturally scalable to massive parallel systems as per (Nair et al., 2015). We believe that, beyond our specific implementation, randomized value functions represent a promising alternative to dithering for exploration. Bootstrapped DQN practically combines efficient generalization with exploration for complex nonlinear value functions.

References

Abbasi-Yadkori, Yasin and Szepesvári, Csaba. Bayesian optimal control of smoothly parameterized systems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2015.

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*, 2012.

Bickel, Peter J and Freedman, David A. Some asymptotic theory for the bootstrap. *The Annals of Statistics*, pp. 1196–1217, 1981.

Blundell, Charles, Cornebise, Julien, Kavukcuoglu, Koray, and Wierstra, Daan. Weight uncertainty in neural networks. *ICML*, 2015.

Brafman, Ronen I. and Tennenholtz, Moshe. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

Burnetas, Apostolos N and Katehakis, Michael N. Optimal adaptive policies for markov decision processes. *Mathematics of Operations Research*, 22(1):222–255, 1997.

Dann, Christoph and Brunskill, Emma. Sample complexity of episodic fixed-horizon reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2800–2808, 2015.

Efron, Bradley. *The jackknife, the bootstrap and other resampling plans*, volume 38. SIAM, 1982.

Efron, Bradley and Tibshirani, Robert J. *An introduction to the bootstrap*. CRC press, 1994.

Gal, Yarin and Ghahramani, Zoubin. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

Gopalan, Aditya and Mannor, Shie. Thompson sampling for learning parameterized markov decision processes. In *Proceedings of the 28th Conference on Learning Theory (COLT)*, pp. 861–898, 2015.

Graves, Alex. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pp. 2348–2356, 2011.

Guez, Arthur, Silver, David, and Dayan, Peter. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems*, pp. 1025–1033, 2012.

Guez, Arthur, Heess, Nicolas, Silver, David, and Dayan, Peter. Bayes-adaptive simulation-based search with value function approximation. In *Advances in Neural Information Processing Systems*, pp. 451–459, 2014.

Hernández-Lobato, José Miguel and Adams, Ryan P. Probabilistic backpropagation for scalable learning of bayesian neural networks. *ICML*, 2015.

Jaksch, Thomas, Ortner, Ronald, and Auer, Peter. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

Kakade, Sham. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.

Kearns, Michael J. and Singh, Satinder P. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3): 209–232, 2002.

Kingma, Diederik P, Salimans, Tim, and Welling, Max. Variational dropout and the local reparameterization trick. *arXiv preprint arXiv:1506.02557*, 2015.

Kleiner, Ariel, Talwalkar, Ameet, Sarkar, Purnamrita, and Jordan, Michael I. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(4):795–816, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lai, Tze Leung and Robbins, Herbert. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6 (1):4–22, 1985.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

Mnih, Volodymyr et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Munos, Rémi. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. 2014.

Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, De Maria, Alessandro, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Osband, Ian and Van Roy, Benjamin. Model-based reinforcement learning and the eluder dimension. In *Advances in Neural Information Processing Systems*, pp. 1466–1474, 2014a.

Osband, Ian and Van Roy, Benjamin. Near-optimal reinforcement learning in factored MDPs. In *Advances in Neural Information Processing Systems*, pp. 604–612, 2014b.

Osband, Ian and Van Roy, Benjamin. Bootstrapped thompson sampling and deep exploration. *arXiv preprint arXiv:1507.00300*, 2015.

Osband, Ian, Russo, Daniel, and Van Roy, Benjamin. (More) efficient reinforcement learning via posterior sampling. In *NIPS*, pp. 3003–3011. Curran Associates, Inc., 2013.

Osband, Ian, Van Roy, Benjamin, and Wen, Zheng. Generalization and exploration via randomized value functions. *arXiv preprint arXiv:1402.0635*, 2014.

Owen, Art B, Eckles, Dean, et al. Bootstrapping data arrays of arbitrary order. *The Annals of Applied Statistics*, 6(3):895–927, 2012.

Rubin, Donald B et al. The bayesian bootstrap. *The annals of statistics*, 9(1):130–134, 1981.

Rusmevichientong, Paat and Tsitsiklis, John N. Linearly parameterized bandits. *Math. Oper. Res.*, 35(2):395–411, 2010.

Russo, Daniel and Van Roy, Benjamin. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39 (4):1221–1243, 2014.

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

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Stadie, Bradly C, Levine, Sergey, and Abbeel, Pieter. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Strehl, Alexander L and Littman, Michael L. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*, pp. 856–863. ACM, 2005.

Strens, Malcolm J. A. A bayesian framework for reinforcement learning. In *ICML*, pp. 943–950, 2000.

Sutton, Richard and Barto, Andrew. *Reinforcement Learning: An Introduction*. MIT Press, March 1998.

Teh, Yee Whye, Hasenclever, Leonard, Lienart, Thibaut, Vollmer, Sebastian, Webb, Stefan, Lakshminarayanan, Balaji, and Blundell, Charles. Distributed bayesian learning with stochastic natural-gradient expectation propagation and the posterior server. *arXiv preprint arXiv:1512.09327*, 2015.

Tesauro, Gerald. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Thompson, W.R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

Wang, Tao, Lizotte, Daniel, Bowling, Michael, and Schuurmans, Dale. Bayesian sparse sampling for on-line reward optimization. In *Proceedings of the 22nd international conference on Machine learning*, pp. 956–963. ACM, 2005.

Wang, Ziyu, de Freitas, Nando, and Lanctot, Marc. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Wen, Zheng and Van Roy, Benjamin. Efficient exploration and value function generalization in deterministic systems. In *NIPS*, pp. 3021–3029, 2013.

APPENDICES

A. Uncertainty for neural networks

In this appendix we discuss some of the experimental setup to qualitatively evaluate uncertainty methods for deep neural networks. To do this, we generated twenty noisy regression pairs x_i, y_i with:

$$y_i = x_i + sin(\alpha(x_i + w_i)) + sin(\beta(x_i + w_i)) + w_i$$

where x_i are drawn uniformly from $(0, 0.6) \cup (0.8, 1)$ and $w_i \sim N(\mu = 0, \sigma^2 = 0.03^2)$. We set $\alpha = 4$ and $\beta = 13$. None of these numerical choices were important except to represent a highly nonlinear function with lots of noise and several clear regions where we should be uncertain. We present the regression data together with an indication of the generating distribution in Figure 16.

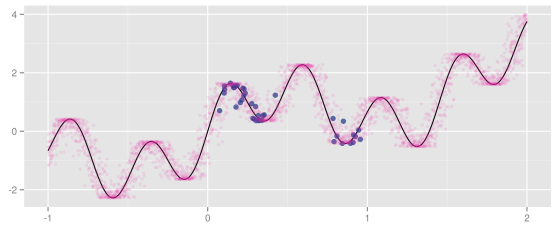


Figure 16: Underlying generating distribution. All our algorithms receive the same blue data. Pink points represent other samples, the mean function is shown in black.

Interestingly, we did not find that using dropout produced satisfying confidence intervals for this task. We present one example of this dropout posterior estimate in Figure 17.

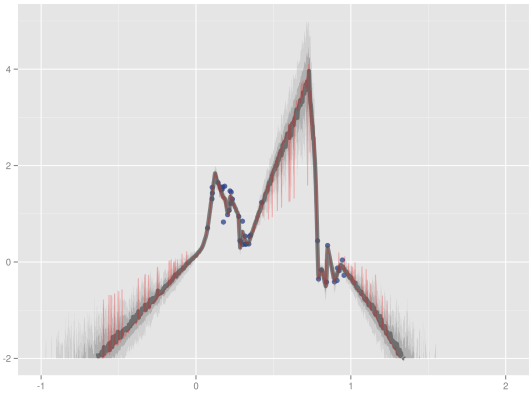


Figure 17: Dropout gives strange uncertainty estimates.

These results are unsatisfactory for several reasons. First, the network extrapolates the mean posterior far outside the range of any actual data for $x = 0.75$. We believe this is because dropout only perturbs locally from a single neural network fit, unlike bootstrap. Second, the posterior samples from the dropout approximation are very spiky and do not look like any sensible posterior sample. Third, the network collapses to almost zero uncertainty in regions with data.

We spent some time altering our dropout scheme to fix this effect, which might be undesirable for stochastic domains and we believed might be an artefact of our implementation. However, after further thought we believe this to be an effect which you would expect for dropout posterior approximations. In Figure 18 we present a didactic example taken from the author's website (Gal & Ghahramani, 2015).

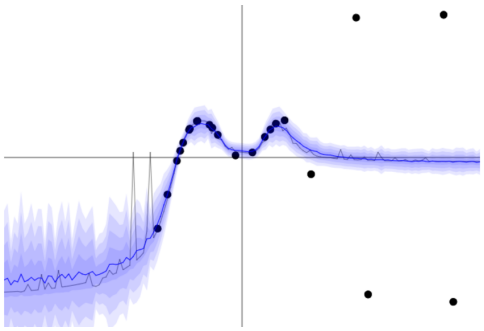


Figure 18: Screenshot from accompanying web demo to (Gal & Ghahramani, 2015). Dropout converges with high certainty to the mean value.

On the right hand side of the plot we generate noisy data with wildly different values. Training a neural network using MSE criterion means that the network will surely converge to the mean of the noisy data. Any dropout samples remain highly concentrated around this mean. By contrast, bootstrapped neural networks may include different subsets of this noisy data and so may produce a more intuitive uncertainty estimates for our settings.

In this paper we focus on the bootstrap approach to uncertainty for neural networks. We like its simplicity, connections to established statistical methodology and empirical good performance. However, the key insights of this paper is the use of deep exploration via randomized value functions. This is compatible with any approximate posterior estimator for deep neural networks. We believe that this area of uncertainty estimates for neural networks remains an important area of research in its own right.

Bootstrapped uncertainty estimates for the Q-value functions have another crucial advantage over dropout which does not appear in the supervised problem. Unlike random dropout masks trained against random target networks, our implementation of bootstrap DQN trains against its own *temporally consistent* target network. This means that our bootstrap estimates (in the sense of (Efron, 1982)), are able to "bootstrap" (in the TD sense of (Sutton & Barto, 1998)) on their own estimates of the long run value. This is important to quantify the long run uncertainty over Q and drive deep exploration.

B. Experiments for deep exploration

B.1. Bootstrap methodology

A naive implementation of bootstrapped DQN builds up K complete networks with K distinct memory buffers. This method is parallelizable up to many machines, however we wanted to produce an algorithm that was efficient even on a single machine. To do this, we implemented the bootstrap heads in a single larger network, like Figure 1 but without any shared network. We implement bootstrap by masking each episode of data according to $w_1, .., w_K \sim \text{Ber}(p)$.

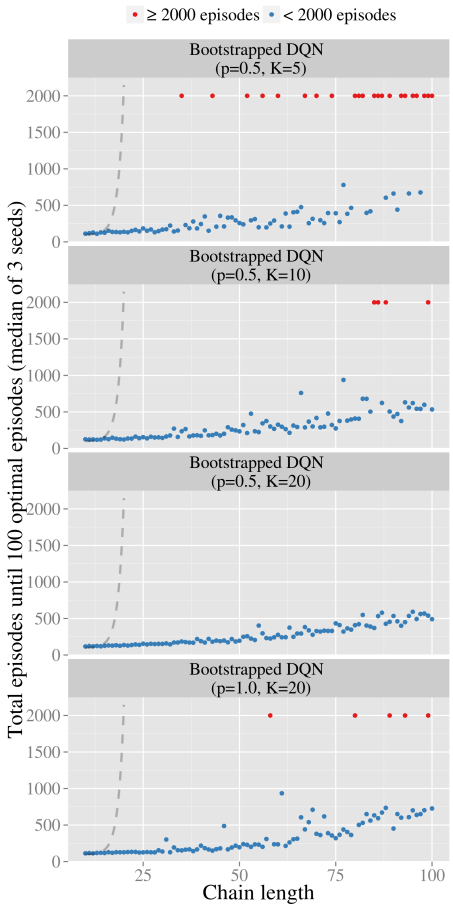


Figure 19: Bootstrapped DQN performs well even with small number of bootstrap heads K or high probability of sharing p.

In Figure 19 we demonstrate that bootstrapped DQN can implement deep exploration even with relatively small values of K. However, the results are more robust and scalable with larger K. We run our experiments on the example from Figure 4. Surprisingly, this method is even effective with $p = 1$ and complete data sharing between heads. This degenerate full sharing of information turns out to be remarkably efficient for training large and deep neural networks. We discuss this phenomenon more in Appendix C.

Generating good estimates for uncertainty is not enough for efficient exploration. In Figure 20 we see that other methods trained with the same network architecture are totally ineffective at implementing deep exploration. The ϵ-greedy policy follows just one Q-value estimate. We allow this policy to be evaluated without dithering. The ensemble policy is trained exactly as per bootstrapped DQN except at each stage the algorithm follows the policy which is majority vote of the bootstrap heads. Thompson sampling is the same as bootstrapped DQN except a new head is sampled every timestep, rather than every episode.

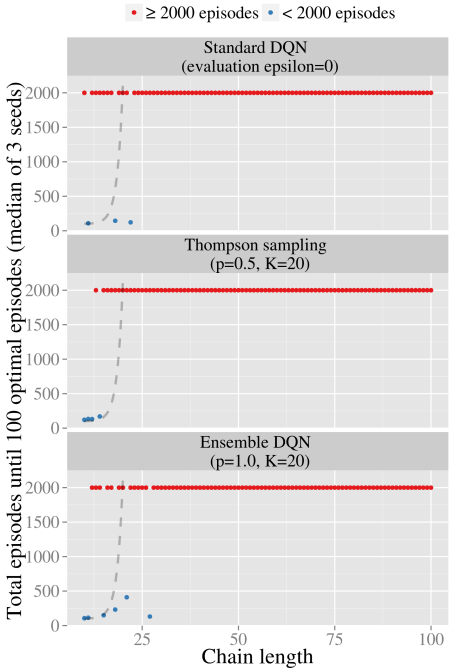


Figure 20: Shallow exploration methods do not work.

We can see that only bootstrapped DQN demonstrates efficient and deep exploration in this domain.

B.2. Comparison to efficient tabular methods

For the example in Figure 6 we attempted to display our performance compared to several benchmark tabula rasa approaches to exploration. There are many other algorithms we could have considered, but for a short paper we chose to focus against the most common approach (ϵ-greedy) the pre-eminent optimistic approach (UCRL2) and posterior sampling (PSRL).

Other common heuristic approaches, such as optimistic initialization for Q-learning can be tuned to work well on this domain, however the precise parameters are sensitive to the underlying MDP[6]. To make a general-purpose version of this heuristic essentially leads to optimistic algorithms. Famous optimistic algorithms like Rmax (Braf-

[6]Further, it is difficult to extend the idea of optimistic initialization with function generalization, especially for deep neural networks.

man & Tennenholtz, 2002), E3 (Kearns & Singh, 2002) or MBIE (Strehl & Littman, 2005) can be thought of as earlier approaches to optimistic exploration that are generally superseded by UCRL2. Since UCRL2 is originally designed for infinite-horizon MDPs, we use the natural adaptation of this algorithm, which has state of the art guarantees in finite horizon MDPs as well (Dann & Brunskill, 2015).

Figure 7 displays the empirical regret of these algorithms together with bootstrapped DQN on the example from Figure 6. It is somewhat disconcerting that UCRL2 appears to incur linear regret, but it is proven to satisfy near-optimal regret bounds. Actually, as we show in Figure 21, the algorithm produces regret which scales very similarly to its established bounds (Jaksch et al., 2010). Similarly, even for this tiny problem size, the recent analysis that proves a near optimal sample complexity in fixed horizon problems (Dann & Brunskill, 2015) only guarantees that we will have fewer than 10^{10} $\epsilon = 1$ suboptimal episodes. While these bounds may be acceptable in worst case $\tilde{O}(\cdot)$ scaling, they are not of much practical use.

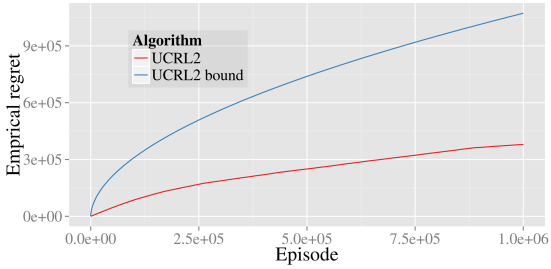


Figure 21: The regret bounds for UCRL2 are near-optimal in $\tilde{O}(\cdot)$, but they are still not very practical.

B.3. How/why does this give deep exploration?

In some ways bootstrapped DQN, with K fixed heads that generate alternative policies, resembles evolutionary or randomized policy algorithms. However, there are several key differences. Bootstrapped DQN exhibits deep exploration which allows it to learn exponentially faster. Purely policy-based methods typically require learning time that scales with the number of *policies*. For an MDP with S states, A actions and H timesteps this is $O(A^{SH})$, or 2^{N^2} in the example from Figure 4 and potentially even worse than dithering strategies $O(2^N)$. If we modified Figure 4 to remove the small reward then, prior to observing the reward there would be zero signal for *any* policy gradient algorithm. This can be formalized to a proof that policy gradients may take $O(2^{N^2})$ episodes to learn (Osband et al., 2014). In practice, the small reward makes these algorithms even worse, since it draws the policy *away* from the optimal. Bootstrapped DQN is a very different algorithm, we now present some more intuition for why.

Imagine that after L episodes the agent has explored all actions in states $1 \leq s \leq n-1 < N$ and has only once taken an action from $s = n < N$. Since the agent has never been to $s = n+1$ then we know that it has never taken action right in $s = n$. For n, L large we can imagine that all K bootstrap heads will estimate $Q^*(s, a)$ based upon the observed optimal policy to head left $a = 1$. However, if the networks generalize in a diverse way to unknown states (due to random initialization, different target networks and different datasets) then they should produce different estimates for $Q_k(n, 2)$.

As long as one head k imagines $Q_k(n, 2) > Q_k(n, 1)$ then, through the TD bootstrapping in head k this incentive for deep exploration will propagate throughout the entire chain. If this estimate of $Q_k(n, 2)$ is high enough, then when head k is chosen it will produce a policy that leads the agent to state $n + 1$. The expected time for these estimates at n to propagate to at least one head grows gracefully in n, even for relatively small K. This is very different from policy-based or evolutionary algorithms.

It is important to note that for our implementations we rely on the random initialization of deep neural networks as some kind of prior to induce diversity. Unlike supervised learning where all networks fit the same data, the target networks mean that small differences at initialization are prone to become bigger as they refit to unique TD errors. This is effective for our experimental setting, but this will not work in all situations. In stochastic environments or with rescaled rewards it may be necessary to augment the bootstrap heads with artificial prior data to maintain diversity in generalization (Osband & Van Roy, 2015).

B.4. One-hot features

In Figure 22 we include the mean performance of bootstrapped DQN with one-hot feature encodings. We found that, using these features, bootstrapped DQN learned the optimal policy for most seeds, but was somewhat less robust than the thermometer encoding. Two out of ten seeds failed to learn the optimal policy within 2000 episodes, this is presented in Figure 22.

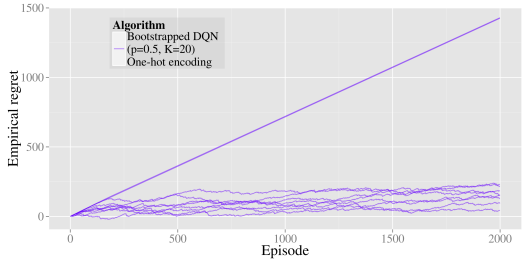


Figure 22: Bootstrapped DQN also performs well with one-hot features, but learning is less robust.

C. Experiments for Atari

C.1. Experimental setup

We use the same 49 Atari games as (Mnih, 2015) for our experiments. Each step of the agent corresponds to four steps of the emulator, where the same action is repeated, the reward values of the agents are clipped between -1 and 1 for stability. We evaluate our agents and report performance based upon the raw scores.

The convolutional part of the network used is identical to the one used in (Mnih, 2015). The input to the network is 4x84x84 tensor with a rescaled, grayscale version of the last four observations. The first convolutional (conv) layer has 32 filters of size 8 with a stride of 4. The second conv layer has 64 filters of size 4 with stride 2. The last conv layer has 64 filters of size 3. We split the network beyond the final layer into $K = 10$ distinct heads, each one is fully connected and identical to the single head of DQN (Mnih, 2015). This consists of a fully connected layer to 512 units followed by another fully connected layer to the Q-Values for each action. The fully connected layers all use Rectified Linear Units(ReLU) as a non-linearity. We normalize gradients $1/K$ that flow from each head.

We trained the networks with RMSProp with a momentum of 0.95 and a learning rate of 0.00025 as in (Mnih, 2015). The discount was set to $\gamma = 0.99$, the number of steps between target updates was set to $\tau = 10000$ steps. We trained the agents for a total of 50m steps per game, which corresponds to 200m frames. The agents were every 1m frames, for evaluation in bootstrapped DQN we use an ensemble voting policy. The experience replay contains the 1m most recent transitions. We update the network every 4 steps by randomly sampling a minibatch of 32 transitions from the replay buffer to use the exact same minibatch schedule as DQN. For training we used an ϵ-greedy policy with ϵ being annealed linearly from 1 to 0.01 over the first 1m timesteps.

C.2. Gradient normalization in bootstrap heads

Most literature in deep RL for Atari focuses on learning the best single evaluation policy, with particular attention to whether this above or below human performance (Mnih, 2015). This is unusual for the RL literature, which typically focuses upon cumulative or final performance.

Bootstrapped DQN makes significant improvements to the cumulative rewards of DQN on Atari, as we display in Figure 15, while the peak performance is much more We found that using bootstrapped DQN without gradient normalization on each head typically learned even faster than our implementation with rescaling $1/K$, but it was somewhat prone to premature and suboptimal convergence. We present an example of this phenomenon in Figure 23.

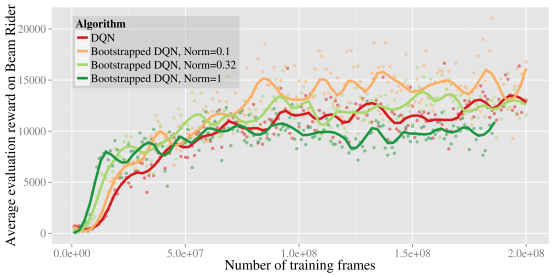


Figure 23: Normalization fights premature convergence.

We found that, in order to better the benchmark "best" policies reported by DQN, it was very helpful for us to use the gradient normalization. However, it is not entirely clear whether this represents an improvement for all settings. In Figures 24 and 25 we present the cumulative rewards of the same algorithms on Beam Rider.

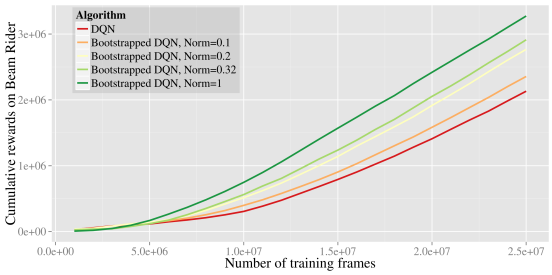


Figure 24: Normalization does not help cumulative rewards.

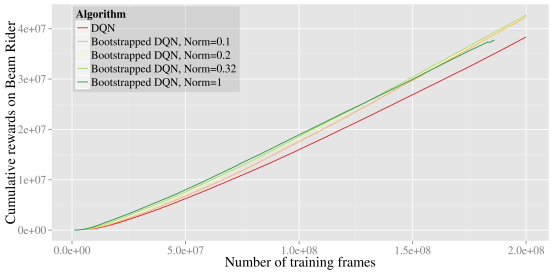


Figure 25: Even over 200m frames the importance of exploration dominates the effects of an inferior final policy.

Where an RL system is deployed to learn with real interactions, cumulative rewards present a better measure for performance. In these settings the benefits of gradient normalization are less clear. However, even with normalization $1/K$ bootstrapped DQN significantly outperforms DQN in terms of cumulative rewards. This is reflected most clearly in Figure 15 and Table 2.

C.3. Sharing data in bootstrap heads

As we report in Section 6 our results on Atari are reported for a degenerate masking of data $p = 1$[7]. In this setting all network heads share all the data, so they are not actually a traditional bootstrap at all. This is different from the regression task in Section 2, where bootstrapped data was essential to obtain meaningful uncertainty estimates. We have several theories for why the networks maintain significant diversity even without data bootstrapping in this setting. We build upon the intuition of Appendix B.3.

First, they all train on different target networks. This means that even when facing the same (s, a, r, s') datapoint this can still lead to drastically different Q-value updates. Second, Atari is a deterministic environment, any transition observation is the unique correct datapoint for this setting. Third, the networks are deep and initialized from different random values so they will likely find quite diverse generalization even when they agree on given data. Finally, since all variants of DQN take many many frames to update their policy, it is likely that even using $p = 0.5$ they would still populate their replay memory with identical datapoints. This means using $p = 1$ to save on minibatch passes seems like a reasonable compromise and it doesn't seem to negatively affect performance too much in this setting. More research is needed to examine exactly where/when this data sharing is important.

C.4. A note on Montezuma's revenge

The game Montezuma's revenge has posed a major challenge for most RL algorithms. This environment is conceptually similar to Figure 4 for a large N and without the small reward. This algorithm requires deep exploration since rewards are extremely sparse. Based on the arguments in this paper we would hope that bootstrap DQN provides significant benefits here. Figure 26 shows the learning curves on this game.

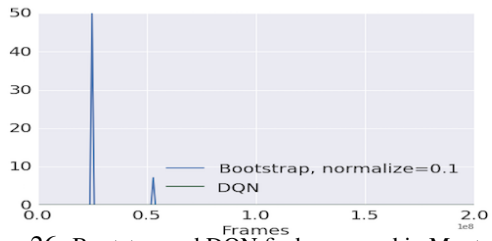


Figure 26: Bootstrapped DQN finds a reward in Montezuma.

Importantly, bootstrapped DQN successfully finds a reward after fewer than 25M frames. DQN with ϵ-greedy exploration fails to find any reward after 200M frames. However, bootstrapped DQN does not successfully learn from this signal to reach sustainably good performance. This is not entirely surprising, since training on random minibatches from $1M$ previous transitions the actual amount of training signal from this reward will be close to zero. However, if we paired this exploration with an aggressive prioritized replay (Schaul et al., 2015) this could provide an avenue towards efficient learning in Montezuma's revenge.

[7]We do this for computational reasons, however in a distributed system we can implement bootstrapped DQN in a naturally parallel manner (Nair et al., 2015).

C.5. Results tables

In Table 1 the average score achieved by the agents during the most successful evaluation period, compared to human performance and a uniformly random policy. DQN is our implementation of DQN with the hyperparameters specified above, using the double Q-Learning update.(Van Hasselt et al., 2015). We find that peak final performance is similar under bootstrapped DQN to previous benchmarks.

To compare the benefits of exploration via bootstrapped DQN we benchmark our performance against the most similar prior work on incentivizing exploration in Atari (Stadie et al., 2015). To do this, we compute the AUC-100 measure specified in this work. We present these results in Table 2 compare to their best performing strategy as well as their implementation of DQN. Importantly, bootstrapped DQN outperforms this prior work significantly.

	Random	Human	Bootstrapped DQN	DDQN	Nature
Alien	227.8	7127.7	2436.6	**4007.7**	3069
Amidar	5.8	1719.5	1272.5	**2138.3**	739.5
Assault	222.4	742.0	**8047.1**	6997.9	3359
Asterix	210.0	8503.3	**19713.2**	17366.4	6012
Asteroids	719.1	47388.7	1032.0	**1981.4**	1629
Atlantis	12850.0	29028.1	**994500.0**	767850.0	85641
Bank Heist	14.2	753.1	**1208.0**	1109.0	429.7
Battle Zone	2360.0	37187.5	**38666.7**	34620.7	26300
Beam Rider	363.9	16926.5	**23429.8**	16650.7	6846
Bowling	23.1	160.7	60.2	**77.9**	42.4
Boxing	0.1	12.1	**93.2**	90.2	71.8
Breakout	1.7	30.5	**855.0**	437.0	401.2
Centipede	2090.9	12017.0	4553.5	4855.4	**8309**
Chopper Command	811.0	7387.8	4100.0	5019.0	**6687**
Crazy Climber	10780.5	35829.4	**137925.9**	137244.4	114103
Demon Attack	152.1	1971.0	82610.0	**98450.0**	9711
Double Dunk	-18.6	-16.4	**3.0**	-1.8	-18.1
Enduro	0.0	860.5	**1591.0**	1496.7	301.8
Fishing Derby	-91.7	-38.7	**26.0**	19.8	-0.8
Freeway	0.0	29.6	**33.9**	33.4	30.3
Frostbite	65.2	4334.7	2181.4	**2766.8**	328.3
Gopher	257.6	2412.5	**17438.4**	13815.9	8520
Gravitar	173.0	3351.4	286.1	**708.6**	306.7
Hero	1027.0	30826.4	**21021.3**	20974.2	19950
Ice Hockey	-11.2	0.9	**-1.3**	-1.7	-1.6
Jamesbond	29.0	302.8	**1663.5**	1120.2	576.7
Kangaroo	52.0	3035.0	**14862.5**	14717.6	6740
Krull	1598.0	2665.5	8627.9	**9690.9**	3805
Kung Fu Master	258.5	22736.3	**36733.3**	36365.7	23270
Montezuma Revenge	0.0	4753.3	**100.0**	0.0	0
Ms Pacman	307.3	6951.6	2983.3	**3424.6**	2311
Name This Game	2292.3	8049.0	11501.1	**11744.4**	7257
Pong	-20.7	14.6	**20.9**	**20.9**	18.9
Private Eye	24.9	69571.3	**1812.5**	158.4	1788
Qbert	163.9	13455.0	15092.7	**15209.7**	10596
Riverraid	1338.5	17118.0	12845.0	**14555.1**	8316
Road Runner	11.5	7845.0	**51500.0**	49518.4	18257
Robotank	2.2	11.9	66.6	**70.6**	51.6
Seaquest	68.4	42054.7	9083.1	**19183.9**	5286
Space Invaders	148.0	1668.7	2893.0	**4715.8**	1976
Star Gunner	664.0	10250.0	55725.0	**66091.2**	57997
Tennis	-23.8	-8.3	0.0	**11.8**	-2.5
Time Pilot	3568.0	5229.2	9079.4	**10075.8**	5947
Tutankham	11.4	167.6	214.8	**268.0**	186.7
Up N Down	533.4	11693.2	**26231.0**	19743.5	8456
Venture	0.0	1187.5	212.5	239.7	**380**
Video Pinball	0.0	17667.9	**811610.0**	685911.0	42684
Wizard Of Wor	563.5	4756.5	6804.7	**7655.7**	3393
Zaxxon	32.5	9173.3	11491.7	**12947.6**	4977

Table 1: Maximal evaluation Scores achieved by agents

We now compare our method against the results in (Stadie et al., 2015). In this paper they introduce a new measure of performance called AUC-100, which is something similar to normalized cumulative rewards up to 20 million frames. Table 2 displays the results for our reference DQN and bootstrapped DQN as Boot-DQN. We reproduce their reference results for DQN as DQN* and their best performing algorithm, Dynamic AE. We also present bootstrapped DQN without head rescaling as Boot-DQN+.

	DQN*	Dynamic AE	DQN	Boot-DQN	Boot-DQN+
Alien	0.15	0.20	0.23	0.23	**0.33**
Asteroids	0.26	0.41	0.29	0.29	**0.55**
Bank Heist	0.07	0.15	0.06	0.09	**0.77**
Beam Rider	0.11	0.09	0.24	0.46	**0.79**
Bowling	0.96	**1.49**	0.24	0.56	0.54
Breakout	0.19	0.20	0.06	0.16	**0.52**
Enduro	0.52	0.49	1.68	**1.85**	1.72
Freeway	0.21	0.21	0.58	0.68	**0.81**
Frostbite	0.57	0.97	0.99	**1.12**	0.98
Montezuma Revenge	0.00	0.00	0.00	0.00	0.00
Pong	0.52	0.56	-0.13	0.02	**0.60**
Qbert	0.15	0.10	0.13	0.16	**0.24**
Seaquest	0.16	0.17	0.18	0.23	**0.44**
Space Invaders	0.20	0.18	0.25	0.30	**0.38**
Average	0.29	0.37	0.35	0.41	**0.62**

Table 2: AUC-100 for different agents compared to (Stadie et al., 2015)

We see that, on average, both bootstrapped DQN implementations outperform Dynamic AE, the best algorithm from previous work. The only game in which Dynamic AE produces best results is Bowling, but this difference in Bowling is dominated by the implementation of DQN* vs DQN. Bootstrapped DQN still gives over 100% improvement over its relevant DQN baseline. Overall it is clear that Boot-DQN+ (bootstrapped DQN without rescaling) performs best in terms of AUC-100 metric. Averaged across the 14 games it is over 50% better than the next best competitor, which is bootstrapped DQN with gradient normalization.

However, in terms of peak performance over 200m frames Boot-DQN generally reached higher scores. Boot-DQN+ sometimes plateaud early as in Figure 23. This highlights an important distinction between evaluation based on best learned policy versus cumulative rewards, as we discuss in Appendix C.2. Bootstrapped DQN displays the biggest improvements over DQN when doing well during learning is important.