Deep Exploration via Bootstrapped DQN

Ian Osband[1,2], **Charles Blundell**[2], **Alexander Pritzel**[2], **Benjamin Van Roy**[1]
[1]Stanford University, [2]Google DeepMind
{iosband, cblundell, apritzel}@google.com, bvr@stanford.edu

Abstract

Efficient exploration remains a major challenge for reinforcement learning (RL). Common dithering strategies for exploration, such as ϵ-greedy, do not carry out temporally-extended (or deep) exploration; this can lead to exponentially larger data requirements. However, most algorithms for statistically efficient RL are not computationally tractable in complex environments. Randomized value functions offer a promising approach to efficient exploration with generalization, but existing algorithms are not compatible with nonlinearly parameterized value functions. As a first step towards addressing such contexts we develop *bootstrapped DQN*. We demonstrate that bootstrapped DQN can combine deep exploration with deep neural networks for exponentially faster learning than any dithering strategy. In the Arcade Learning Environment bootstrapped DQN substantially improves learning speed and cumulative performance across most 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 in order 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. Further, since any action may have long term consequences, 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 [9]. However, most of these are designed for Markov decision processes (MDPs) with small finite state spaces, while others require solving computationally intractable planning tasks [8]. 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 statistically inefficient strategies for exploration [12] or even no exploration at all [23]. We review related literature in more detail in Section 4.

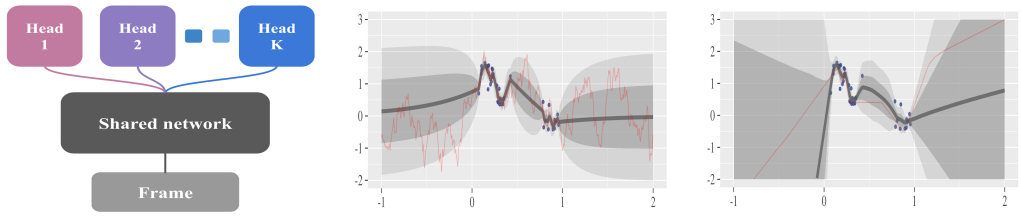
Common dithering strategies, such as ϵ-greedy, approximate the value of an action by a single number. Most of the time they pick the action with the highest estimate, but sometimes they choose another action at random. In this paper, we consider an alternative approach to efficient exploration inspired by Thompson sampling. These algorithms have some notion of uncertainty and instead maintain a *distribution* over possible values. They explore by randomly select a policy according to the probability it is the optimal policy. Recent work has shown that randomized value functions can implement something similar to Thompson sampling without the need for an intractable exact posterior update. However, this work is restricted to linearly-parameterized value functions [16]. We present a natural

extension of this approach that enables use of complex non-linear generalization methods such as deep neural networks. We show that the bootstrap with random initialization can produce reasonable uncertainty estimates for neural networks at low computational cost. Bootstrapped DQN leverages these uncertainty estimates for efficient (and deep) exploration. We demonstrate that these benefits can extend to large scale problems that are not designed to highlight deep exploration. Bootstrapped DQN substantially reduces learning times and improves performance across most games. This algorithm is computationally efficient and parallelizable; on a single machine our implementation runs roughly 20% slower than DQN.

2 Uncertainty for neural networks

Deep neural networks (DNN) represent the state of the art in many supervised and reinforcement learning domains [12]. We want an exploration strategy that is statistically computationally efficient together with a DNN representation of the value function. To explore efficiently, the first step 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 [3, 7]. We actually found the simple non-parametric bootstrap with random initialization [5] more effective in our experiments, but the main ideas of this paper would apply with any other approach to uncertainty in DNNs.

The bootstrap princple is to approximate a population distribution by a sample distribution [6]. 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 even comes with theoretical guarantees [2]. In Figure 1a 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 and represents a single bootstrap sample $\psi(\tilde{D})$. The shared network learns a joint feature representation across all the data, which can provide significant computational advantages at the cost of lower diversity between heads. This type of bootstrap can be trained efficiently in a single forward/backward pass; it can be thought of as a data-dependent dropout, where the dropout mask for each head is fixed for each data point [19].



(a) Shared network architecture (b) Gaussian process posterior (c) Bootstrapped neural nets

Figure 1: Bootstrapped neural nets can produce reasonable posterior estimates for regression.

Figure 1 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, 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 [7]. Further details are provided in Appendix A.

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 from state s_t, action a_t, reward r_t and new state 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 [12]. 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 [25] modifies the target y_t^Q and helps further[1]:

$$y_t^Q \leftarrow r_t + \gamma \max_a Q\big(s_{t+1}, \arg\max_a Q(s_{t+1},a;\theta_t);\theta^-\big). \quad (2)$$

Bootstrapped DQN modifies DQN to approximate a *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 adaptation of the Thompson sampling heuristic to RL that allows for temporally extended (or deep) exploration [21, 13].

We implement this algorithm efficiently by building up $K \in \mathbb{N}$ bootstrapped estimates of the Q-value function in parallel as in Figure 1a. 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. In order to keep track of which data belongs to which bootstrap head we store flags $w_1,..,w_K \in \{0,1\}$ indicating which heads are privy to which data. We approximate a bootstrap sample by selecting $k \in \{1,..,K\}$ uniformly at random and following Q_k for the duration of that episode. We present a detailed algorithm for our implementation of bootstrapped DQN in Appendix B.

4 Related work

The observation that temporally extended exploration is necessary for efficient reinforcement learning is not new. 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[8]. Many successful RL applications focus on generalization and planning but address exploration only via inefficient exploration [12] or even none at all [23]. However, such exploration strategies can be highly inefficient.

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 proposed for finite-armed bandits [11], but the principle has been extended successfully across bandits with generalization and tabular RL [9]. Except for particular deterministic contexts [27], OFU methods that lead to efficient RL in complex domains have been computationally intractable. The work of [20] 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 D and find that bootstrapped DQN offers a significant improvement over previous methods.

Perhaps the oldest heuristic for balancing exploration with exploitation is given by Thompson sampling [24]. 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. To apply the Thompson sampling principle to RL, an agent should sample a value function from its posterior. Naive applications of Thompson sampling to RL which resample every timestep can be extremely

[1] In this paper we use the DDQN update for all DQN variants unless explicitly stated.

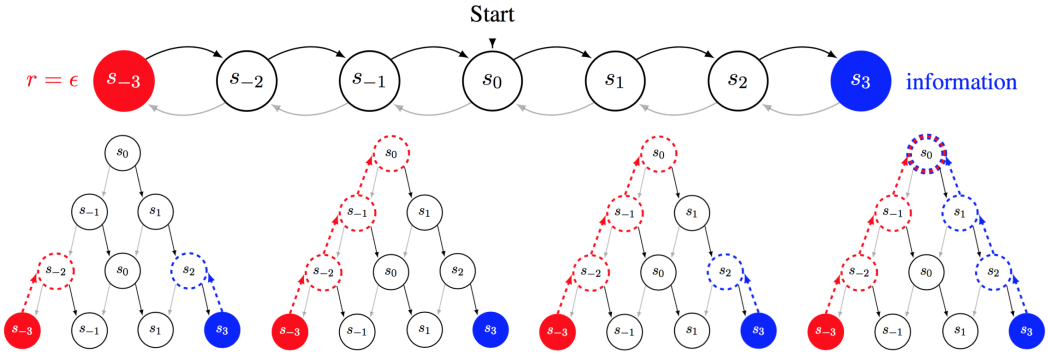
inefficient. The agent must also commit to this sample for several time steps in order to achieve deep exploration [21, 8]. The algorithm PSRL does exactly this, with state of the art guarantees [13, 14]. However, this algorithm still requires solving a single known MDP, which will usually be intractable 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, RLSVI 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 [16]. 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 [10]. 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 2 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 explore (d) RL+deep explore

Figure 2: 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. These environments can be described by chains of length $N > 3$ in Figure 3. Each episode of interaction lasts $N + 9$ steps after which point the agent resets to the initial state s_2. These are toy problems intended to be expository rather than entirely realistic. Balancing a well known and mildly successful strategy versus an unknown, but potentially more rewarding, approach can emerge in many practical applications.

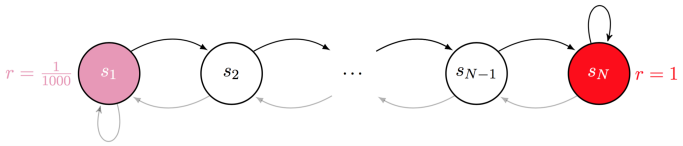


Figure 3: Scalable environments that requires deep exploration.

These environments may be described by a finite tabular MDP. However, we consider algorithms which interact with the MDP only through raw pixel features. We consider two feature mappings $\phi_{\text{1hot}}(s_t) := (\mathbb{1}\{x = s_t\})$ and $\phi_{\text{therm}}(s_t) := (\mathbb{1}\{x \leq s_t\})$ in $\{0,1\}^N$. We present results for ϕ_{therm}, which worked better for all DQN variants due to better generalization, but the difference was relatively small - see Appendix C. Thompson DQN is the same as bootstrapped DQN, but resamples every timestep. Ensemble DQN uses the same architecture as bootstrapped DQN, but with an ensemble policy.

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 4, together with a conservative lower bound of $99 + 2^{N-11}$ on the expected time to learn for any shallow exploration strategy [16]. Only bootstrapped DQN demonstrates a graceful scaling to long chains which require deep exploration.

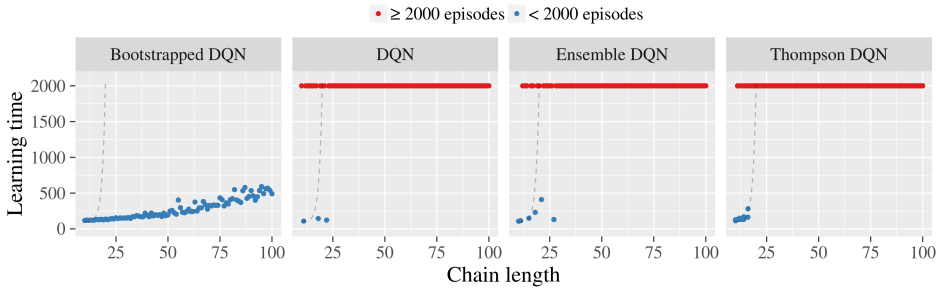


Figure 4: Only Bootstrapped DQN demonstrates deep exploration.

5.2 How does bootstrapped DQN drive deep exploration?

Bootstrapped DQN explores in a manner similar to the provably-efficient algorithm PSRL [13] but it uses a bootstrapped neural network to approximate a posterior sample for the value. Unlike PSRL, bootstrapped DQN directly samples a value function and so does not require further planning steps. This algorithm is similar to RLSVI, which is also provably-efficient [16], but with a neural network instead of linear value function and bootstrap instead of Gaussian sampling. The analysis for the linear setting suggests that this nonlinear approach will work well so long as the distribution $\{Q^1, .., Q^K\}$ remains *stochastically optimistic* [16], or at least as spread out as the "correct" posterior. 數學上如何解讀?

Bootstrapped DQN relies upon random initialization of the network weights as a prior to induce diversity. Surprisingly, we found this initial diversity was enough to maintain diverse generalization to new and unseen states for large and deep neural networks. This is effective for our experimental setting, but will not work in all situations. In general it may be necessary to maintain some more rigorous notion of "prior", potentially through the use of artificial prior data to maintain diversity [15]. One potential explanation for the efficacy of simple random initialization is that unlike supervised learning or bandits, where all networks fit the same data, each of our Q^k heads has a unique target network. This, together with stochastic minibatch and flexible nonlinear representations, means that even small differences at initialization may become *bigger* as they refit to unique TD errors.

這裡到更新時使用相同的樣本，但head會有各自的target NN。在使用不同初始的參數，兼

Bootstrapped DQN does *not* require that any single network Q^k is initialized to the correct policy of "right" at every step, which would be exponentially unlikely for large chains N. For the algorithm to be successful in this example we only require that the networks generalize in a diverse way to the actions they have never chosen in the states they have not visited very often. Imagine that, in the example above, the network has made it as far as state $\tilde{N} < N$, but never observed the action right $a = 2$. As long as one head k imagines $Q(\tilde{N}, 2) > Q(\tilde{N}, 2)$ then TD bootstrapping can propagate this signal back to $s = 1$ through the target network to drive deep exploration. The expected time for these estimates at n to propagate to at least one head grows gracefully in n, even for relatively small K, as our experiments show. We expand upon this intuition with a video designed to highlight *how* bootstrapped DQN demonstrates deep exploration `https://youtu.be/e3KuV_d0EMk`. We present further evaluation on a difficult stochastic MDP in Appendix C.

6 Arcade Learning Environment

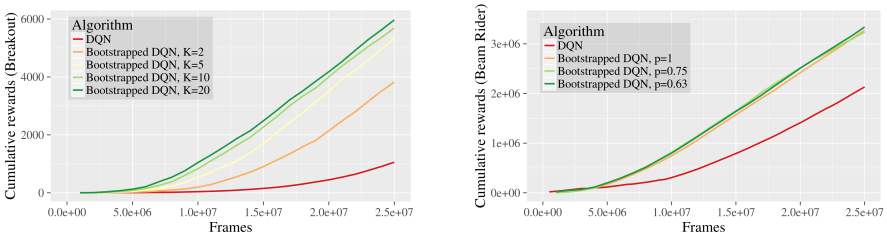
We now evaluate our algorithm across 49 Atari games on the Arcade Learning Environment [1]. Importantly, and unlike the experiments in Section 5, these domains are not specifically designed to showcase our algorithm. In fact, many Atari games are structured so that small rewards always indicate part 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 [25] for our network architecture and benchmark our performance against their algorithm. Our network structure is identical to the convolutional structure of DQN [12] except we split 10 separate bootstrap heads after the convolutional layer as per Figure 1a. Recently, several authors have provided architectural and algorithmic improvements to DDQN [26, 18]. 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 D.

6.1 Implementing bootstrapped DQN at scale

We now examine how to generate online bootstrap samples for DQN in a computationally efficient manner. We focus on 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? We make significant compromises in order to maintain computational cost comparable to DQN.

Figure 5a 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 $K = 10$.



(a) Number of bootstrap heads K. (b) Probability of data sharing p.

Figure 5: Examining the sensitivities of bootstrapped DQN.

The shared network architecture allows us to train this combined network via backpropagation. 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. See Appendix D for more details.

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

To implement an online bootstrap we use an independent Bernoulli mask $w_1,..,w_K \sim \text{Ber}(p)$ for each head in each episode[3]. These flags are stored in the memory replay buffer and identify which heads are trained on which data. However, when trained using a shared minibatch the algorithm will also require an effective $1/p$ more iterations; this is undesirable computationally. Surprisingly, we found the algorithm performed similarly irrespective of p and all outperformed DQN, as shown in Figure 5b. This is strange and we discuss this phenomenon in Appendix D. However, 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[4].

6.2 Efficient exploration in Atari

We find that Bootstrapped DQN drives efficient exploration in several Atari games. For the same amount of game experience, bootstrapped DQN generally outperforms DQN with ϵ-greedy exploration. Figure 6 demonstrates this effect for a diverse selection of games.

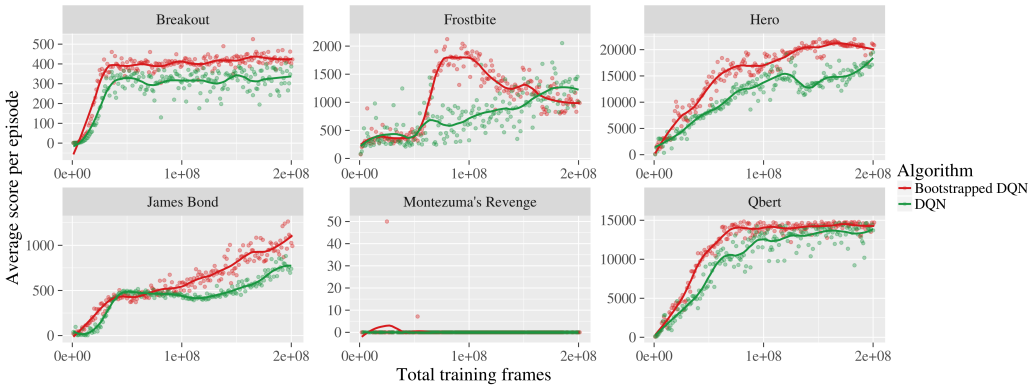


Figure 6: Bootstrapped DQN drives more efficient exploration.

On games where DQN performs well, bootstrapped DQN typically performs better. Bootstrapped DQN does not reach human performance on Amidar (DQN does) but does on Beam Rider and Battle Zone (DQN does not). To summarize this improvement in learning time we consider the number of frames required to reach human performance. If bootstrapped DQN reaches human performance in $1/x$ frames of DQN we say it has improved by x. Figure 7 shows that Bootstrapped DQN typically reaches human performance significantly faster.

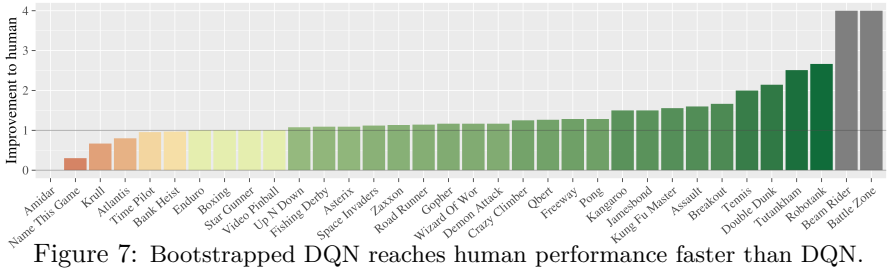


Figure 7: Bootstrapped DQN reaches human performance faster than DQN.

On most games where DQN does not reach human performance, bootstrapped DQN does not solve the problem by itself. On some challenging Atari games where deep exploration is conjectured to be important [25] our results are not entirely successful, but still promising. In Frostbite, bootstrapped DQN reaches the second level much faster than DQN but network instabilities cause the performance to crash. In Montezuma's Revenge, bootstrapped DQN reaches the first key after 20m frames (DQN never observes a reward even after 200m frames) but does not properly learn from this experience[5]. Our results suggest that improved exploration may help to solve these remaining games, but also highlight the importance of other problems like network instability, reward clipping and temporally extended rewards.

[3] p=0.5 is double-or-nothing bootstrap [17], p=1 is ensemble with no bootstrapping at all.
[4] Our implementation K=10, p=1 ran with less than a 20% increase on wall-time versus DQN.
[5] An improved training method, such as prioritized replay [18] may help solve this problem.

6.3 Overall performance

Bootstrapped DQN is able to learn much faster than DQN. Figure 8 shows that bootstrapped DQN also improves upon the final score across most games. However, the real benefits to *efficient* exploration mean that bootstrapped DQN outperforms DQN by orders of magnitude in terms of the *cumulative* rewards through learning (Figure 9. In both figures we normalize performance relative to a fully random policy. The most similar work to ours presents several other approaches to improved exploration in Atari [20] they optimize for AUC-20, a normalized version of the cumulative returns after 20m frames. According to their metric, averaged across the 14 games they consider, we improve upon both base DQN (0.29) and their best method (0.37) to obtain 0.62 via bootstrapped DQN. We present these results together with results tables across all 49 games in Appendix D.4.

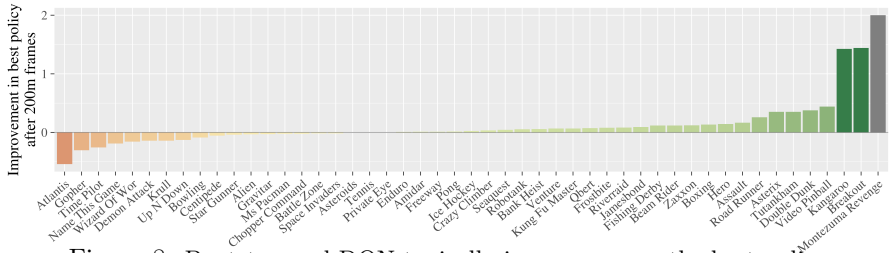


Figure 8: Bootstrapped DQN typically improves upon the best policy.

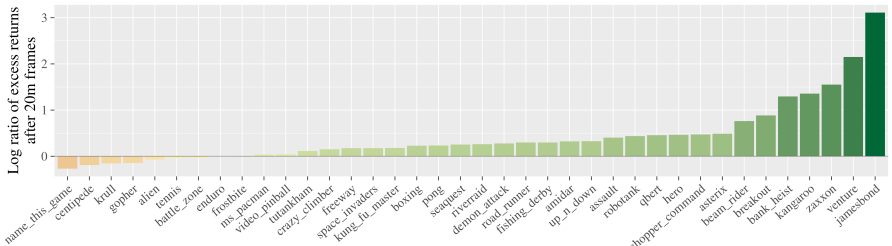


Figure 9: Bootstrapped DQN improves cumulative rewards by orders of magnitude.

6.4 Visualizing bootstrapped DQN

We now present some more insight to how bootstrapped DQN drives deep exploration in Atari. In each game, although each head $Q^1, .., Q^{10}$ learns a high scoring policy, the policies they find are quite distinct. In the video `https://youtu.be/Zm2KoT82O_M` we show the evolution of these policies simultaneously for several games. Although each head performs well, they each follow a unique policy. By contrast, ϵ-greedy strategies are almost indistinguishable for small values of ϵ and totally ineffectual for larger values. We believe that this deep exploration is key to improved learning, since diverse experiences allow for better generalization.

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. 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. In the video `https://youtu.be/0jvEcC5JvGY` we visualize this ensemble policy across several games. We find that the uncertainty in this policy is surprisingly interpretable: all heads agree at clearly crucial decision points, but remain diverse at other less important steps.

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 is computationally tractable and also naturally scalable to massive parallel systems. 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] Malcolm J. A. Strens. A bayesian framework for reinforcement learning. In *ICML*, pages 943–950, 2000.

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

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

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

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

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

[27] Zheng Wen and Benjamin Van Roy. Efficient exploration and value function generalization in deterministic systems. In *NIPS*, pages 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 10.

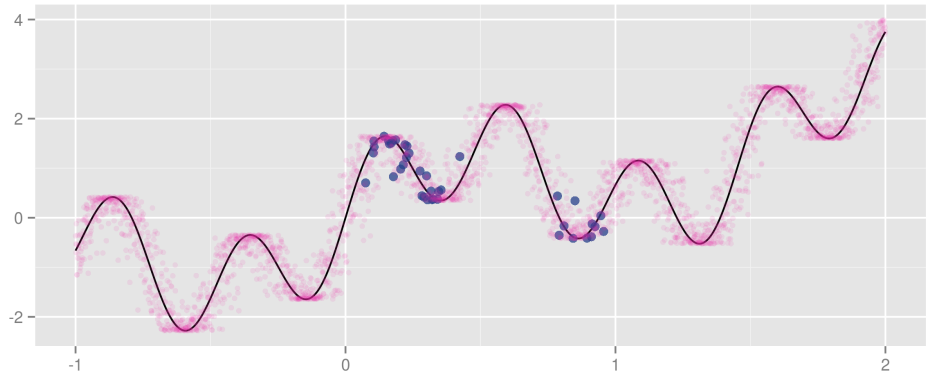
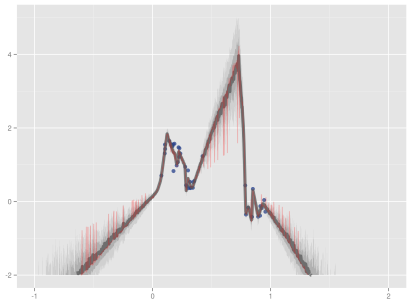
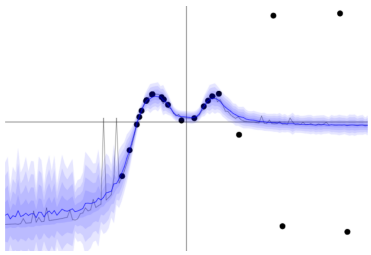


Figure 10: Underlying generating distribution. All our algorithms receive the same blue data. Pink points represent other samples, the mean function is shown in black.

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 11a.



(a) Dropout gives strange uncertainty estimates.



(b) Screenshot from accompanying web demo to [7]. Dropout converges with high certainty to the mean value.

Figure 11: Comparing the bootstrap to dropout uncertainty for neural nets.

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 11b we present a didactic example taken from the author's website [7].

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. Note this isn't necessarily a failure of dropout to approximate a Gaussian process posterior, but this artefact could be shared by any homoskedastic posterior. The authors of [7] propose a heteroskedastic variant which can help, but does not address the fundamental issue that for large networks trained to convergence all dropout samples may converge to every single datapoint... even the outliers.

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 [5]), are able to "bootstrap" (in the TD sense of [22]) 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 Bootstrapped DQN implementation

Algorithm 1 gives a full description of Bootstrapped DQN. It captures two modes of operation where either k neural networks are used to estimate the Q_k-value functions, or where one neural network with k heads is used to estimate k Q-value functions. In both cases, as this is largely a parameterisation issue, we denote the value function networks as Q, where Q_k is output of the kth network or the kth head.

A core idea to the full bootstrapped DQN algorithm is the bootstrap mask m_t. The mask m_t decides, for each value function Q_k, whether or not it should train upon the experience generated at step t. In its simplest form m_t is a binary vector of length K, masking out or including each value function for training on that time step of experience (i.e., should it receive gradients from the corresponding $(s_t, a_t, r_{t+1}, s_{t+1}, m_t)$ tuple). The masking distribution M is responsible for generating each m_t. For example, when M yields m_t whose components are independently drawn from a bernoulli distribution with parameter 0.5 then this corresponds to the double-or-nothing bootstrap [17]. On the other hand, if M yields a mask m_t with all ones, then the algorithm reduces to an ensemble method. Poisson masks $M_t[k] \sim \text{Poi}(1)$ provides the most natural parallel with the standard non-parameteric boostrap since $\text{Bin}(N, 1/N) \to \text{Poi}(1)$ as $N \to \infty$. Exponential masks $M_t[k] \sim \text{Exp}(1)$ closely resemble the standard Bayesian nonparametric posterior of a Dirichlet process [15].

Periodically, the replay buffer is played back to update the parameters of the value function network Q. The gradients of the kth value function Q_k for the tth tuple in the replay buffer B, g_t^k is:

$$g_t^k = m_t^k(y_t^Q - Q_k(s_t, a_t; \theta))\nabla_\theta Q_k(s_t, a_t; \theta) \tag{3}$$

where y_t^Q is given by (2). Note that the mask m_t^k modulates the gradient, giving rise to the bootstrap behaviour.

Algorithm 1 Bootstrapped DQN

Input: Value function networks Q with K outputs $\{Q_k\}_{k=1}^K$. Masking distribution M.
Let B be a replay buffer storing experience for training.
for each episode **do**
    Obtain initial state from environment s_0
    Pick a value function to act using $k \sim \text{Uniform}\{1, \ldots, K\}$
    for step $t = 1, \ldots$ until end of episode **do**
        Pick an action according to $a_t \in \arg\max_a Q_k(s_t, a)$
        Receive state s_{t+1} and reward r_t from environment, having taking action a_t
        Sample bootstrap mask $m_t \sim M$
        Add $(s_t, a_t, r_{t+1}, s_{t+1}, m_t)$ to replay buffer B
    end for
end for

C Experiments for deep exploration

C.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 1a but without any shared network. We implement bootstrap by masking each episode of data according to $w_1, .., w_K \sim \text{Ber}(p)$.

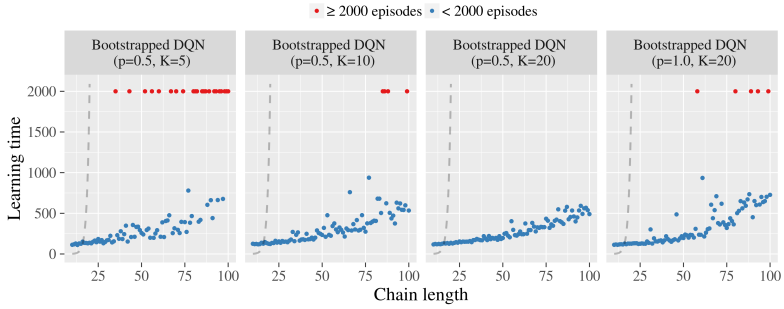


Figure 12: Bootstrapped DQN performs well even with small number of bootstrap heads K or high probability of sharing p.

In Figure 12 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 3. 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 D.

Generating good estimates for uncertainty is not enough for efficient exploration. In Figure 13 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.

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

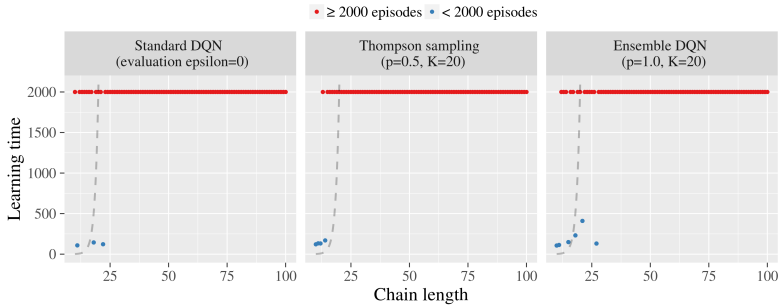


Figure 13: Shallow exploration methods do not work.

C.2 A difficult stochastic MDP

Figure 4 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 14. 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.

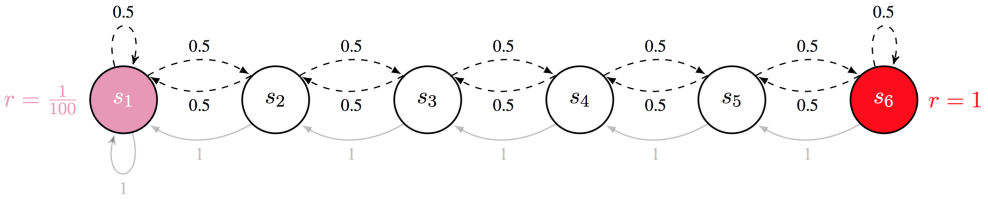
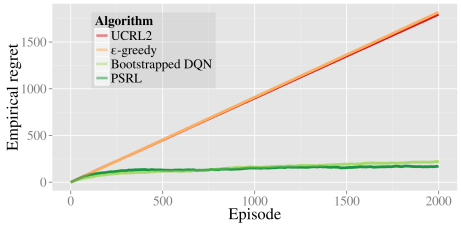
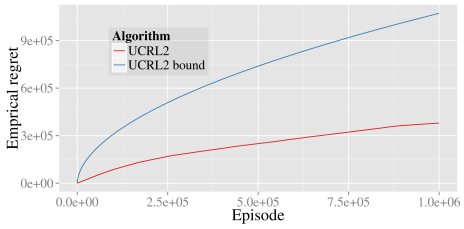


Figure 14: A stochastic MDP that requires deep exploration.

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 [9] and posterior sampling [13]. We discuss the choice of these benchmarks in Appendix C.



(a) Bootstrapped DQN matches efficient tabular RL.



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

Figure 15: Learning and regret bounds on a stochastic MDP.

In Figure 15a 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 15a seems to suggest UCRL2 incurs linear regret, actually it follows its bounds $\tilde{O}(S\sqrt{AT})$ [9] where S is the number of states and A is the number of actions.

For the example in Figure 14 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. 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 [4].

Figure 15a displays the empirical regret of these algorithms together with bootstrapped DQN on the example from Figure 14. 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 15b, the algorithm produces regret which scales very similarly to its established bounds [9]. Similarly, even for this tiny problem size, the recent analysis that proves a near optimal sample complexity in fixed horizon problems [4] 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.

C.3 One-hot features

In Figure 16 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 16.

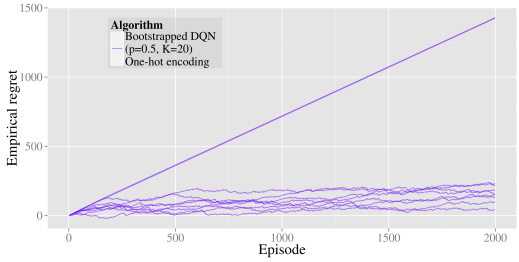


Figure 16: Bootstrapped DQN also performs well with one-hot features, but learning is less robust.

D Experiments for Atari

D.1 Experimental setup

We use the same 49 Atari games as [12] 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 [12]. The input to the network is 4x84x84 tensor with a rescaled, grayscale version of the last four observations.

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

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 [12]. 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 [12]. 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.

D.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 [12]. 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 9, 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 17.

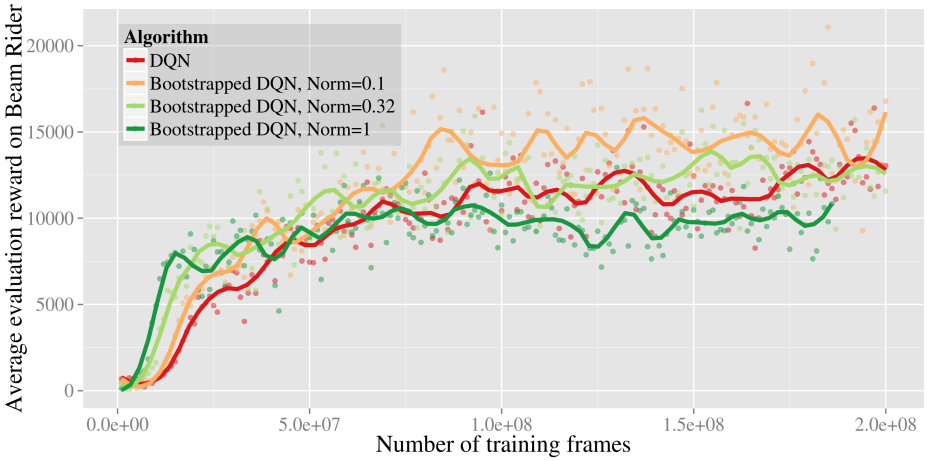
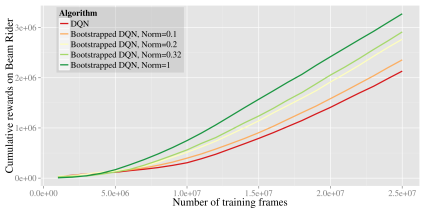


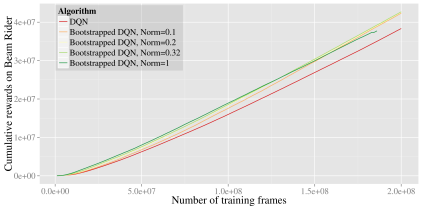
Figure 17: Normalization fights premature convergence.

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 18a and 18b we present the cumulative rewards of the same algorithms on Beam Rider.

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 9 and Table 2.



(a) Normalization does not help cumulative rewards.



(b) Even over 200m frames the importance of exploration dominates the effects of an inferior final policy.

Figure 18: Planning, learning and exploration in RL.

D.3 Sharing data in bootstrap heads

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 Section 5.2.

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.

D.4 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.[25]. 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 [20]. 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 [20]. 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 [20]

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 17. This highlights an important distinction between evaluation based on best learned policy versus cumulative rewards, as we discuss in Appendix D.2. Bootstrapped DQN displays the biggest improvements over DQN when doing well during learning is important.