EFFICIENT EXPLORATION THROUGH BAYESIAN DEEP Q-NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Bayesian Deep Q-Network (BDQN), a practical Thompson sampling based Reinforcement Learning (RL) Algorithm. Thompson sampling allows for targeted exploration in high dimensions through posterior sampling, but is usually computationally expensive. We address this limitation by introducing uncertainty only at the output layer of the network through a Bayesian Linear Regression (BLR) model, which can be trained with fast closed-form updates and its samples can be drawn efficiently through the Gaussian distribution. We apply our method to a wide range of Atari Arcade Learning Environments. Since BDQN carries out more efficient exploration, it is able to reach higher rewards substantially faster than a key baseline, DDQN.

1 INTRODUCTION

Designing algorithms that achieve an optimal trade-off between exploration and exploitation is the primary goal of reinforcement learning (RL). However, targeted exploration in high dimensional spaces is a challenging problem in RL. Recent works on deep RL mostly employ simple exploration strategies such as ε-greedy, where the agent chooses the optimistic action with probability $(1-\varepsilon)$, otherwise, uniformly at random picks one of the available actions. Due to this uniform sampling, the ε-greedy method scales poorly with the dimensionality of state and action spaces.

Recent work has considered scaling exploration strategies to large domains (Tang et al., 2016; Bellemare et al., 2017; Osband et al., 2016). Several of these papers have focused on employing optimism-under-uncertainty approaches, which essentially rely on computing confidence bounds over different actions, and acting optimistically with respect to that uncertainty.

An alternative to optimism-under-uncertainty is Thompson Sampling (TS) (Thompson, 1933), where we draw samples from a posterior distribution over models. It is a Bayesian framework where we start with a prior distribution and compute the posterior beliefs based on the collected data through interaction with the environment. We then maximize the expected reward under the sampled belief. Posterior sampling can provide more targeted exploration since it can trade off uncertainty with the expected return of actions. In contrast, the ε-greedy strategy is indifferent to uncertainty of the actions and the expected rewards of sub-optimistic ones (set of actions excluding the optimistic action).

There is compelling evidence that Thompson Sampling can provide better results than optimism-under-uncertainty approaches, but so far this has only been demonstrated on smaller scale domains (Osband et al., 2013). There has been relatively little work on scaling Thompson Sampling to large state spaces. The primary difficulty in implementing Thompson sampling is the difficulty of sampling from general posterior distributions. Prior efforts in this space have generally required extremely expensive computations (e.g.(Ghavamzadeh et al., 2015)).

In this paper, we derive a practical Thompson sampling framework, termed as Bayesian deep Q-networks (BDQN), where we sample from the posterior on the set of Q-functions. BDQN is computationally efficient since it incorporates uncertainty only at the output layer, in the form of a Bayesian linear regression model. Due to linearity and by choosing a Gaussian prior, we derive a closed-form analytical update to the posterior distribution over Q functions. We can also draw samples efficiently from the Gaussian distribution.

We test BDQN on a wide range of Arcade Learning Environment Atari games (Bellemare et al., 2013)(Brockman et al., 2016) against a strong baseline DDQN (Van Hasselt et al., 2016). Aside from simplicity and popularity of DDQN, BDQN and DDQN share the same architecture (except for the last layer), and follow same target objective. These are the main reasons we chose DDQN for our comparisons.

In table. 1 we see significant gains for BDQN over DDQN. BDQN is able to learn significantly faster and reach higher rewards due to more efficient exploration. The evidence of this is further seen from the fact that we are able to train BDQN with much higher learning rates compared to DDQN. This suggests that BDQN is able to learn faster and reach better scores.

These promising results suggest that BDQN can further benefit from additional modifications that were done to DQN, e.g. Prioritized Experience Replay (Schaul et al., 2015), Dueling approach (Wang et al., 2015), A3C (Mnih et al., 2016) etc. This is because BDQN only changes that exploration strategy of DQN, and can easily accommodate additional improvements to DQN.

Game	Amidar	Assault	Pong	Asteroids	BattleZone	Atlantis
No. of Frames	100M	100M	100M	100M	50M	20M
$\frac{\text{BDQN}}{\text{DDQN}}$	557%	396%	129%	2517%	281%	8785%

Table 1: Percentage of improvement in score

2 Related Work

The complexity of the exploration-exploitation trade-off has been vastly investigated in RL. Auer (2002) addresses this question for multi-armed bandit problems where regret guarantees are provided. Jaksch et al. (2010) investigates the regret analyses in MDPs where exploration happens through the optimal policy of optimistic model, a well-known Optimism in Face of Uncertainty (OFU) principle where a high probability regret upper bound is guaranteed. Azizzadenesheli et al. (2016) deploys OFU in order to propose the high probability regret upper bound for Partially Observable MDPs (POMDPs) and finally, Bartók et al. (2014) tackles a general case of partial monitoring games and provides minimax regret guarantee which is polynomial in certain dimensions of the problem.

In the OFU based methods, the exploration is under a policy which is optimal for the most optimistic setting while the OFU based methods might fail to provide an efficient exploration (Lattimore & Szepesvari, 2016). Agrawal & Goyal (2012); Gopalan & Mannor (2015); Agrawal & Jia (2017) deploy TS for the exploration where the frequentist guarantees are provided while the Bayesian analyses are studied by Osband et al. (2014; 2013); Abbasi-Yadkori & Szepesvári (2015).

Even through theoretical RL addresses the complexity of these trade-offs, these problems are still prominent in empirical reinforcement learning research (Mnih et al., 2015; Jiang et al., 2016; Abel et al., 2016). On the empirical side, the prominent recent success in the video games has sparked a flurry of research interest. Following the success of Deep RL on Atari games (Mnih et al., 2015) and the board game Go (Silver et al., 2016), many researchers have begun exploring practical applications of deep reinforcement learning (DRL). Some investigated applications include, robotics (Levine et al., 2016), energy management (Night, 2016), and self-driving cars (Shalev-Shwartz et al., 2016). Among the mentioned literature, the prominent exploration strategy for Deep RL agent has been ε-greedy.

For the first time in large scale practical applications, Lipton et al. (2016) used posterior sampling based exploration in dialogue systems, employing a variational based method in order to estimate the posterior distribution over the weights of the network. Recently Osband et al. (2016) introduced using multiple heads of a deep neural network to represent an approximate posterior computed by bootstrapping. However, here, they did not estimate a true posterior since all collected data was used to update all estimated Q-functions, which meant that the diversity between the estimated Q-functions is mainly due to differences in the initialization weights. Also, the explorations are still combined with uniform ε exploration. Moreover, the authors were not able to show superior performance over uniformly sampled DQNs and DDQN. Drop-out, as a randomized exploration method is proposed

by Gal & Ghahramani (2016) but Osband et al. (2016) argues about its non-informative uncertainty estimate. These results motivate us to investigate TS for practical classes of RL problems.

3 PRELIMINARIES

An infinite horizon λ-discounted MDP M is a tuple $\langle \mathcal{X}, \mathcal{A}, T, R \rangle$, with state space $\mathcal{X}$, action space $\mathcal{A}$, and the transition kernel T, accompanied with reward function of $\mathcal{R}$ where $0 < \lambda \leq 1$. At each time step t, the environment is at a state x_t, called current state, where the agent needs to make a decision a_t under its policy. Given the current state and action, the environment stochastically proceed to a successor state x_{t+1} under probability distribution $T(X_{t+1}|x_t, a_t) := \mathbb{P}(X_{t+1}|x_t, a_t)$ and provides a stochastic reward r_t with mean of $\mathbb{E}[r|x = x_t, a = a_t] = R(x_t, a_t)$. The agent objective is to optimize the overall expected discounted reward over its policy $\pi := \mathcal{X} \to \mathcal{A}$, a stochastic mapping from states to actions, $\pi(a|x) := \mathbb{P}(a|x)$.

$$\eta^* = \eta(\pi^*) = \max_{\pi} \eta(\pi) = \lim_{N \to \infty} \mathbb{E}_{\pi} \left[\sum_{t=0}^{N} \lambda^t r_t \right] \tag{1}$$

The expectation in Eq. 1 is with respect to the randomness in the distribution of initial state, transition probabilities, stochastic rewards, and policy where η^*, π^* are optimal return and optimal policy, respectively. Let $Q_\pi(x, a)$ denote the average discounted reward under policy π starting off from state x and taking action a in the first place.

$$Q_\pi(x, a) := \lim_{N \to \infty} \mathbb{E}_{\pi} \left[\sum_{t=0}^{N} \lambda^t r_t | x_0 = x, a_0 = a \right]$$

For a given policy π and Markovian assumption of the model, we can rewrite the equation for the Q functions as follows:

$$Q_\pi(x, a) = R(x, a) + \lambda \sum_{x', a'} T(x'|x, a) \pi(a'|x') Q_\pi(x', a') \tag{2}$$

To find the optimal policy, one can solve the Linear Programing problem in Eq. 1 or follow the corresponding Bellman equation Eq. 2 where both of optimization methods turn to

$$Q^*(x, a) = R(x, a) + \lambda \sum_{x'} T(x'|x, a) \max_{a'} Q^*(x', a')$$

where $Q^*(x, a) = Q_{\pi^*}(x, a)$ and the optimal policy is a deterministic mapping from state to actions in $\mathcal{A}$, i.e. $x \to arg \max_a Q^*(x, a)$. Since in RL, we do not know the transition kernel and the reward function in advance and we can not solve the posed Bellman equation directly. In order to tackle this problem, (Antos et al., 2008) studies the property of minimizing the Bellman residual of a given Q-function

$$\mathcal{L}(Q) = \mathbb{E}_{\pi} \left[(Q(x, a) - r - \lambda Q(x', a'))^2 \right] \tag{3}$$

Where the tuple (x, a, r, x', a') consists of consecutive samples under behavioral policy π. Furthermore, (Mnih et al., 2015) carries the same idea, and introduce Deep Q-Network (DQN) where the Q-functions are parameterized by a DNN. To improve the quality of Q function, they use back propagation on loss $\mathcal{L}(Q)$. In the following we descibe the setting used in DDQN. In order to reduce the bias of the estimator, they introduce target network Q^{target} and target value $y = r + \lambda Q^{target}(x', \hat{a})$ where $\hat{a} = \arg\max_{a'} Q(x', a')$ with a new loss $\mathcal{L}(Q, Q^{target})$

$$\mathcal{L}(Q, Q^{target}) = \mathbb{E}_{\pi} \left[(Q(x, a) - y)^2 \right] \tag{4}$$

Minimizing this regression loss, and respectably its estimation $\widehat{\mathcal{L}}(Q, Q^{target})$, matches the Q to the target y and provides an unbiased estimator of the target. Once in a while, the algorithm sets Q^{target} network to Q network and peruse the regression with the new target value.

Algorithm 1 BDQN

Initialize parameter sets θ, θ^{target}, W, W^{target}, and Cov using a normal distribution.
Initialize replay buffer and set counter $= 0$
for episode = 1 to inf **do**
  Initialize x_1 to the initial state of the environment
  for $t =$ to the end of episode **do**
    if count mod $T^{sample} = 0$ **then**
      sample $W \sim \mathcal{N}\left(W^{target}, Cov\right)$
    end if
    Select action $a_t = argmax_{a'} \left[W^\top \phi_\theta(x_t)\right]_{a'}$
    Execute action a_t in environment, observing reward r_t and successor state x_{t+1}
    Store transition (x_t, a_t, r_t, x_{t+1}) in replay buffer
    Sample a random minibatch of transitions (x_τ , a_τ, r_τ, $x_{\tau+1}$) from replay buffer
    $y_\tau \leftarrow \left\{ \begin{array}{ll} r_\tau, & \text{for terminal } x_{\tau+1} \\ r_\tau + \left[W^{target^\top} \phi_{\theta^{target}}(x_{\tau+1})\right]_{\hat{a}}; \hat{a} = argmax_{a'} \left[W^\top \phi_\theta(x_{\tau+1})\right]_{a'} & \text{for non-terminal } x_{\tau+1} \end{array} \right\}$
    $\theta \leftarrow \theta - \eta \cdot \nabla_\theta (y_\tau - \left[W^\top \phi_\theta(x_\tau)\right]_{a_\tau})^2$
    if count mod $T^{target} = 0$ **then**
      set $\theta^{target} \leftarrow \theta$
    end if
    if count mod $T^{Bayes\ target} = 0$ **then**
      Update W^{target} and Cov
    end if
    count = count + 1
  end for
end for

4 BAYESIAN DEEP Q-NETWORKS

In this work, we propose a Bayesian method to approximate the Q-function and match it to the target value. We utilize the DQN architecture and remove its last layer and replace it with Bayesian linear regression (BLR) (Rasmussen & Williams, 2006) to BDQN network. We utilize BLR to efficiently approximate the distortion over the Q-values where the uncertainty over the values is captured. A common assumption in DNN is that the feature representation of the layer before the last layer of the network is suitable for linear classification or linear regression methods (same assumption in DQN).

Let the output feature vector of the layer before the last be represented as $\phi_\theta(x) \in \mathbb{R}^d$, parametrized by θ. Let $\phi_\theta^{target}(x) \in \mathbb{R}^d$ be the feature representation for target value. The Q-functions can be approximated as a linear combination of features in ϕ, i.e for a given pair of state-action $Q(x, a) = \phi_\theta(x)^\top w_a$. Therefore, by deploying BLR, we can write the generative model of the Q-function to its corresponding target value: $y = r + \lambda \phi^{target} w^{target}{}_{\hat{a}}$, for any given state and action pair (x, a) as follows

$$y = Q(x, a) + \epsilon = w_a^\top \phi(x) + \epsilon, \quad \forall x \in \mathcal{X}, a \in \mathcal{A}$$

where scalar ϵ is an independent noise drawn from a Gaussian iid distribution with mean zero and variance σ_ϵ^2, i.e., $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Furthermore, we assume that, for each action $a \in \mathcal{A}$, the vector $w_a \in \mathbb{R}^d$ has a Gaussian prior $\mathcal{N}(0, \sigma^2)$. Therefore, $y|x, a, w_a \sim \mathcal{N}(\phi(x)^\top w_a, \sigma_\epsilon^2)$. Moreover, the distribution of the target value y is as follows:

$$\mathbb{P}(y) = \int_{w_a} \mathbb{P}(y|w_a)\, \mathbb{P}(w_a)\, dw_a \tag{5}$$

Given a dataset $\mathcal{D} = \{x_i, a_i, y_i\}_{i=1}^D$, we construct $|\mathcal{A}|$ disjoint datasets for each action, $\mathcal{D} = \cup_{a \in \mathcal{A}} \mathcal{D}_a$ where $\mathcal{D}_a$ is a set of tuples x_i, a_i, y_i with the action $a_i = a$ and size D_a. Given the data set $\mathcal{D}_a$, we are interested in $\mathbb{P}(w_a|\mathcal{D}_a)$ and $\mathbb{P}(Q(x, a)|\mathcal{D}_a), \forall x \in \mathcal{X}$. Let us construct a matrix $\Phi_a \in \mathbb{R}^{d \times D_a}$, a concatenation of feature column vectors $\{\phi(x_i)\}_{i=1}^{D_a}$, and $\mathbf{y}_a \in \mathbb{R}^{D_a}$, a concatenation of target values

in set $\mathcal{D}_a$. Therefore the posterior distribution is as follows:

$$w_a \sim \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2}\Xi_a\Phi_a\mathbf{y}_a, \Xi_a\right), \ \textit{where} \ \Xi_a = \left(\frac{1}{\sigma_\epsilon^2}\Phi_a\Phi_a^\top + \frac{1}{\sigma^2}\mathbb{1}\right)^{-1} \tag{6}$$

and since the prior and likelihood are conjugate of each other we have

$$Q(x,a)|\mathcal{D}_a \sim \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2}\phi(x)^\top\Xi_a\Phi_a\mathbf{y}_a, \phi(x)^\top\Xi_a\phi(x)\right) \tag{7}$$

where $\mathbb{1} \in \mathbb{R}^d$ is a identity matrix. The expression in Eqs. 6, 7 give the posterior distribution over weights w_a and the function Q respectively. As TS suggests, for the exploration, we exploit the expression in Eq. 6. For all actions, we set w_a^{target} as the mean of posterior distribution over w_a. For each action, we sample a wight vector w_a in order to have samples of mean Q-value. Then we act optimally with respect the sampled means

$$a_{\text{TS}} = \arg\max_a w_a^\top \phi_\theta(x). \tag{8}$$

Let $W = \{w_a\}_{a=1}^{|\mathcal{A}|}$, respectively $W^{target} = \{w_a^{target}\}_{a=1}^{|\mathcal{A}|}$, and $Cov = \{\Xi_a\}_{a=1}^{|\mathcal{A}|}$. In BDQN, the agent interacts with the environment through applying the actions proposed by TS, i.e. a_{TS}. We utilize a notion of replay buffer where the agent stores its recent experiences. The agent draws $W \sim \mathcal{N}\left(W^{target}, Cov\right)$ (abbreviation for sampling of each action separately) every T^{sample} steps and act optimally with respect to drawn weights. During the inner loop of the algorithm, we draw a minibatch of data from replay buffer and use loss

$$(y_\tau - \left[W^\top\phi_\theta(x_\tau)\right]_{a_\tau})^2 \tag{9}$$

where

$$y_\tau := r_\tau + \left[W^{target^\top}\phi_{\theta^{target}}(x_{\tau+1})\right]_{\hat{a}}; \hat{a} = argmax_{a'}\left[W^\top\phi_\theta(x_{\tau+1})\right]_{a'} \tag{10}$$

in order to update the weights of network: $\theta \leftarrow \theta - \eta \cdot \nabla_\theta(y_\tau - \left[W^\top\phi_\theta(x_\tau)\right]_{a_\tau})^2$.

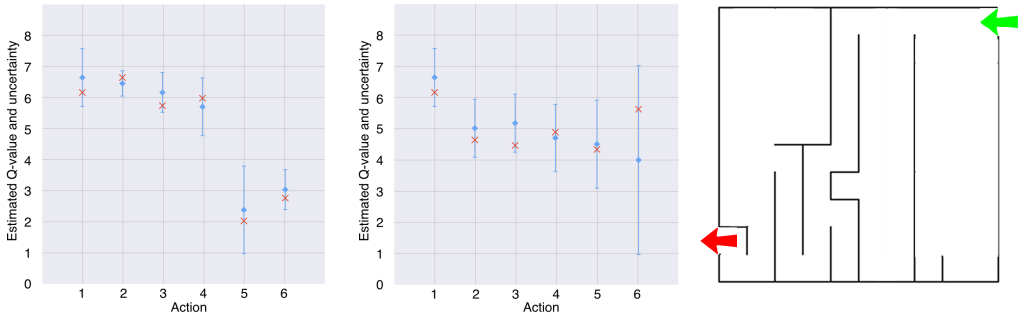
In order to update the target value, we update θ^{target} to θ every T^{target} steps. With the period of $T^{Bayes\ target}$ the agent updates its posterior distribution using larger minibatch of data drawn from replay buffer and sample W with respect to updated posterior. Thus, we train BDQN end-to-end from scratch. Algorithm 1 gives the full description of Bayes DQN.

TS vs ε-greedy: The benefit of TS can be restated using the expression in Eq. 7. Let us assume at a given time point, the environment is at state x and the empirical estimate of Q-function for an action a, $Q(x,a)$, is relatively lower than many of other actions. Further, let us consider the situation where the agent is certain about value of that action Fig. 1*(a)* (actions 5 or 6). In this configuration, exploring the action a might not advance the exploration-exploitation trade-off since the agent is quite sure about the low return of the action a and rather to choose those actions with higher rewards with relatively higher uncertainties.

In this situation, under TS strategy, the chance that the action a gets explored is low and TS randomizes mostly on a set of actions with higher rewards with relatively higher uncertainties (e.g. actions 1,2,3,4). The TS does not pick the action as frequent as other sub-optimistic action until its uncertainty due to the lack of samples increases. On the other hand, ε-greedy based strategies choose the most optimistic action with the probability of $1-\varepsilon$ (most of the time) and for their exploration phase, they choose the action a as frequent as other actions.

Now, let us consider a scenario where the agent is uncertain about the action a, e.g. bigger uncertainty in covariance matrix, and it has low estimated Q-value Fig. 1*(b)*. In this situation, the action a is more likely to be picked by the TS based agent and the corresponding uncertainty gets reduced. In the contrast, the ε-greedy based algorithm behave same as before and keeps choosing the most optimistic action all the time, except with probability ε randomize uniformly over all the actions.

Thus, the exploration phase of the ε-greedy strategies does not exploit all the valuable information which makes them indifference to the estimated Q-value of sub-optimistic actions while not considering the uncertainty level of any action.



(a) (b) (c)

Figure 1: A cartoon representation of TS vs ε-greedy. The red crosses are the true Q-values, the diamonds are the mean of estimated Q-values with blue intervals as uncertainties (e.g. $c \cdot variance$) *(a)*: Even though the agent is certain about low return of action 6, the ε-greedy strategy chooses action 6 as frequent as other sub-optimistic ones while TS based algorithm, randomizing over all the promising actions (most likely actions $1, 2, 3, 4$). Furthermore, it does not pick this action frequently, until its uncertainty, due to the lack of samples, increases. *(b)*: If an action with the low estimated expected return is accompanied with a high uncertainty, the TS based algorithm starts to choose that action until the algorithm reaches the "mean-value and uncertainty" trade-off. On the other hand, ε-greedy strategy does same as *(a)*. *(c)*: a maze-like deterministic game

As addressed in Mnih et al. (2015), one of the major benefits of deploying function approximation methods in deep RL is the estimation of the Q-value given state-action pair, can generalize to the Q-value of similar state-action pairs, even though they are visited rarely. We expect BDQN to, not only generalize the estimation Q-values over the similar state-action pairs, under the same reasoning as DQN, we expect the uncertainty of similar state-action pairs to generalize as well.

In both DQN and BDQN the target values get updated every T^{target} steps of the game which is usually been set to 10000 steps which means that the target values y get updated with low frequency. Moreover, the target values y's smoothly change over time due to distribution shift in replay buffer and adaptive training of targets. Therefore, an optimistic action given the current target values, e.g. action 1 Fig. 1, may become a sub-optimistic action given updated new target values, e.g. action 2, becomes the optimistic action under new target values. Since, before changing the target values, the ε-greedy strategy oversampled the action 1, the optimistic action, without prioritizing sampling of action 2 over others, given new target values, it does not have an enough samples of action 2 in order to have a better estimate of Q-value at that action.

Another superiority of TS over ε-greedy can be described using Fig. 1*(c)*. Consider an episodic maze-inspired deterministic game, where the agent is placed to the green arrow at the beginning of each round of the game and the goal state is exiting form red arrow. Let us further consider that the shortest path from starting point to the end point has a length of H and the duration of each round is exactly H as well. If the agent reaches the goal state in the H time step, it wins the games and receives a score of 1, otherwise, it receives 0. After each H times step, the agent's location is reset to the starting point. Consider an agent, which is given a set of Q-functions where the true Q-function is within the set and is the most optimistic Q-function in the set as well. In this situation, the optimism based strategy, with no exploration, chooses the true Q-function at first place, reaches to the endpoint for all the rounds. On the other hand, TS randomizes over the Q-functions with high promising returns and relatively low uncertainty, including the true Q-function. When it picks the true Q-function, it increases the posterior probability of this Q-function, since it matches the likelihood. When TS chooses other functions since they predict deterministically wrong values, the posterior update of those functions set zero posteriors to them, therefore, the model will not choose them again, i.e. TS finds the true Q-function very fast. For ε-greedy based agent, even though it chooses the true function right away from the begging (it is the optimistic one) and aims to follow the suggested path by true Q-function, at each time step it randomizes its action whit probity ε. Therefore, it takes exponentially many trials in order to get the score on in this game.

5 EXPERIMENTS

We apply BDQN on a variety of games in the OpenAI-Gym [1](Brockman et al., 2016). As a baseline[2], we run DDQN algorithm and evaluate BDQN on the measures of sample complexity and score.

Network architecture: The input x to the network part of BDQN is $4 \times 84 \times 84$ tensor with a rescaled, gray-scale version of the last four observations. The first convolution layer has 32 filters of size 8 with a stride of 4. The second convolution layer has 64 filters of size 4 with stride 2. The last convolution layer has 64 filters of size 3 followed by a fully connected layers with size 512. We add a BLR layer on top of this.

Choice of hyperparameters: For BDQN, we set the values of W^{target} to mean of the prior distribution over the weights of BLR and draw W from that prior. For the fixed W and W^{target}, we randomly initialize the parameters of network part of BDQN, θ, and trained it using RMSProp, a learning rate of 0.0025, and a momentum of 0.95, inspired by (Mnih et al., 2015) where the discount factor is $\gamma = 0.99$, the number of steps between target updates T^{target} steps, and weights W are re-sampled from their posterior distribution with mean every T^{sample} step. We update the network part of BDQN every 4 steps by randomly sampling a mini-batch of size 32 samples from the replay buffer. We update the posterior distribution of the weight set W every $T^{Bayes\ target}$ using mini-batch of size B, with entries sampled uniformly form replay buffer. We set W^{target} to the posterior mean and the covariances to Cov. The experience replay contains the $1M$ most recent transitions. Further hyperparameters, e.g. no-opt, are equivalent to ones in DQN setting.

BDQN has a few more hyperparameters to tune. For the BLR, we have noise variance σ_ϵ, variance of prior over weights σ, sample size B, posterior update period $T^{Bayes\ target}$, and the posterior sampling period T^{sample}. To fine the first three, we set up a simple hyper parameter search. We used a pretrained DQN model for game of *Assault*, and removed the last fully connected layer in order to have access to its already pretrained feature representation. Then we tried combination of $B = \{T^{target}, 10 \cdot T^{target}\}$, $\sigma = \{1, 0.1, 0.001\}$, and $\sigma_\epsilon = \{1, 10\}$ and test for 1000 episode of the game. We set these parameters to their best $B = 10 \cdot T^{target}, \sigma = 0.001, \sigma = 1$.

The above hyperparameter tuning is cheap since it requires only $O(B)$ number of forward passes. For the remaining parameter, we ran BDQN (with weights randomly initialized) on the same game, *Assault*, for $5M$ time steps, with a set of $T^{Bayes\ target} = \{T^{target}, 10 \cdot T^{target}\}$ and $T^{sample} = \{\frac{T^{target}}{10}, \frac{T^{target}}{100}\}$ where BDQN performed better with choice of $T^{Bayes\ target} = 10 \cdot T^{target}$. For both choices of T^{sample}, it performed almost equal where we chose the higher one. In addition to BLR parameters, we use bigger learning compared to DDQN. We just started off with that learning rate and did not tune for that. We guess, due to the efficient exploration of BDQN, it can learn a better policy in even shorter time, in contrast, we experienced that changing the learning of DDQN may cause a major degradation in in its performance. The proposed hyperparameter search is very simple where the exhaustive hyperparameter search is likely to provide even better performance.

Computational and sample cost comparison: For a given period of game time, the number of backward pass in both BDQN and DQN are the same where for BDQN it is cheaper since it has one layer (the last layer) less than DQN. In the sense of fairness in sample usage, for example in duration of $10 \cdot T^{Bayes\ target} = 100k$, all the layers of both BDQN and DQN, except the last layer, see the same number of samples, but the last layer of BDQN sees 16 times less samples compared to the last layer of DQN. The last layer of DQN for a duration of $100k$, observes $25k = 100k/4$ (4 is backprob period) minibatchs of size 32, which is $16 \cdot 100k$, where the last layer of BDQN just observes samples size of $B = 100k$. As it is mentioned in Alg. 1, to update the posterior distribution, BDQN draws B samples from the replay buffer and needs to compute the feature vector of them. This step of BDQN, gives a superiority to DQN in the sense of speed which is almost 50% faster than

[1]We set the internal frame skip of gym to zero and instead manually compensate it in the code. We detailed the environment setting in our code which due to the blind review process, we are not able to add the link

[2]We also attempted to include another baseline: Bootstrapped DQNs (Osband et al., 2016). Due to lack of implementation details, we tried difference reasonable we were not successful to reproduce their results despite extensive experimentation. Moreover, they admit in the paper that uniformly chosen DQN outperforms their setting.

BDQN. One can easily relax this limitation by parallelizing this step with the main body of BDQN or deploying on-line posterior update methods. All the implementations were performed using the MXNet framework (Chen et al., 2015).

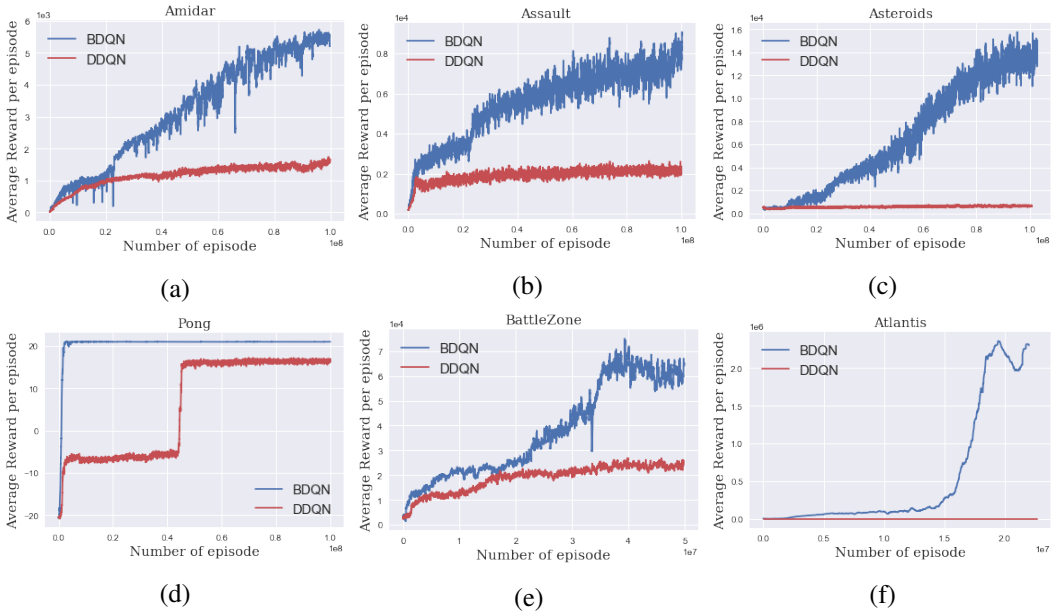


Figure 2: The fast targeted exploration of BDQN

Results: The results are in Fig. 2 and Table. 2. Mostly the focus of the experiments are on sample complexity in Deep-RL, even though, BDQN provides much larger score compared to based line. For example, for game *Atlantis*, DDQN†[3] gives score of $767k$ after $200M$ samples, while BDQN reaches $2.46M$ after $20M$ samples.

We can observe that BDQN can immediately learn a significantly better policy due to its targeted exploration in much shorter period of time. Since BDQN on game *Atlantis* show a big jump around time step $20M$, we ran it two more times in order to make sure it was not just a coincidence. We did the same additional two experiments for the game *Amidar* as well. We observed that the improvements are consistent.

Game	Amidar	Assault	Pong	Asteroids	BattleZone	Atlantis
Frame	100M	100M	100M	100M	50M	20M
BDQN	5.52k	8.84k	21	14.1k	65.2k	2.46M
DDQN	0.99k	2.23k	16.23	0.56k	23.2k	28.0k
DDQN†	2.14k	7.00k	20.9	1.98k	34.62k	767k

Table 2: Comparison of scores

6 Conclusion

In this work we proposed BDQN, a practical TS based RL algorithm which provides targeted exploration in a computationally efficient manner. We demonstrated significantly faster training and enormously better rewards over a strong baseline DDQN. This suggests that BDQN can benefit even

[3]The scores of DDQN† are the reported scores for DDQN after running for 200M samples and are borrowed from Osband et al. (2016). We are not sure about the exact environment set-up, used in their work, therefore the fair comparison might be prohibited.

more from further modifications to DQN such as e.g. Prioritized Experience Replay (Schaul et al., 2015), Dueling approach (Wang et al., 2015), A3C (Mnih et al., 2016) etc. We plan to explore this in future.

REFERENCES

Yasin Abbasi-Yadkori and Csaba Szepesvári. Bayesian optimal control of smoothly parameterized systems. In *UAI*, pp. 1–11, 2015.

David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert E Schapire. Exploratory gradient boosting for reinforcement learning in complex domains. *arXiv preprint arXiv:1603.04119*, 2016.

Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *COLT*, 2012.

Shipra Agrawal and Randy Jia. Posterior sampling for reinforcement learning: worst-case regret bounds. *arXiv*, 2017.

András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 2008.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 2002.

Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. Reinforcement learning of pomdps using spectral methods. In *Proceedings of the 29th Annual Conference on Learning Theory (COLT)*, 2016.

Gábor Bartók, Dean P Foster, Dávid Pál, Alexander Rakhlin, and Csaba Szepesvári. Partial monitoring—classification, regret bounds, and algorithms. *Mathematics of Operations Research*, 2014.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv*, 2015.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, 2016.

Aditya Gopalan and Shie Mannor. Thompson sampling for learning parameterized markov decision processes. In *COLT*, 2015.

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

Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. Contextual decision processes with low bellman rank are pac-learnable. *arXiv*, 2016.

Tor Lattimore and Csaba Szepesvari. The end of optimism? an asymptotic analysis of finite-armed linear bandits. *arXiv*, 2016.

Sergey Levine et al. End-to-end training of deep visuomotor policies. *JMLR*, 2016.

Zachary C Lipton, Jianfeng Gao, Lihong Li, Xiujun Li, Faisal Ahmed, and Li Deng. Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking. *arXiv preprint arXiv:1608.05081*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Will Night. The AI that cut google's energy bill could soon help you. *MIT Tech Review*, 2016.

Ian Osband, Dan Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, 2013.

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

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, 2016.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

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

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv*, 2016.

David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *arXiv*, 2016.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 1933.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.