Efficient Exploration through Bayesian Deep Q-Networks

Kamyar Azizzadenesheli
UCI, Stanford
Email:kazizzad@uci.edu

Emma Brunskill
Stanford
Email:ebrun@cs.stanford.edu

Animashree Anandkumar
Caltech, Amazon
Email:animakumar@gmail.com

***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 games in Arcade Learning Environments. Since BDQN carries out more efficient exploration, it is able to reach higher rewards substantially faster than a key baseline, double deep Q network DDQN.**

I. Introduction

Designing algorithms that achieve an optimal trade-off between exploration and exploitation is one of the primary goals of reinforcement learning (RL). However, targeted and efficient exploration in high dimensional spaces is a challenging problem in RL. Recent advances in deep RL mostly deploy simple exploration strategies such as ε-greedy, where the agent chooses the greedy action, the action with the highest promising return, 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 [1], [2]. 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 [3] is Thompson Sampling (TS) [4], one of the oldest heuristics for multi-arm bandits. TS is a Bayesian approach where one starts with a prior belief (prior distribution) over the environment dynamics and compute the posterior belief by utilizing the data, collected through the interaction with that environment. At the planing time, a sample from the posterior belief is drawn and maximizing the expected return under the sampled belief is deployed. The TS based posterior sampling can provide more targeted exploration since it can trade off uncertainty with the expected return of actions which also has been experienced in the field of psychology [5]. In contrast, the ε-greedy strategy is indifferent to the uncertainty of the actions and the expected rewards of sub-greedy ones (set of actions excluding the greedy action).

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 [6], [7].

In this work, we derive a practical Thompson sampling framework, termed as Bayesian deep Q-networks (BDQN), where we approximate the posterior distribution on the set of Q-functions and sample from this approximated posterior. 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 approximated posterior distribution over Q functions. We can also draw samples efficiently from the Gaussian distribution. Deep RL enjoys the benefits of function approximation when the value function of state-action pairs is a smooth function of states (and actions in the continuous action space) where the learned value of a state-action pair can generalize well to the other similar state-action pairs, to those which haven't been visited [8]. We expect this principle to hold in BDQN, and in additional, we also expect the uncertainty of state-action pairs to generalize as well.

We test BDQN on a wide range of Arcade Learning Environment [9], [10] Atari games against DDQN [11] since, aside from simplicity and popularity of DDQN, BDQN and DDQN share the same architecture, and follow same target objective where BDQN follows Thompson sampling instead of ε-greedy, as is in DDQN.

In Table I we observe significant gains for BDQN over DDQN. BDQN is able to learn significantly faster and reach higher returns 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 master the skills in a shorter amount of time. More detailed results are provided in the latter sections and Fig. 2. In order to observe the significance of BDQN behavior over DDQN, we show the results for a different number of samples. For instance, for the game Pong, we need to show the early stage of the game, $5M$ samples, in order to distinguish the difference in performance Fig. 2. For some games, we show the plots up to the time where the plateaus are observed. Furthermore, we also compare

BDQN with scores of DDQN†[1] which are borrowed from the original DDQN paper [11] where the agent is trained for $200M$ times step and the scores are recorded during the evaluation phase where the ε is set to a small number $(\varepsilon = 0.001)$

These promising results suggest that BDQN can further benefit from additional follow-up modifications made to DQN, e.g. Prioritized Experience Replay [12], Dueling approach [13], A3C [14], safe exploration [15], and etc. This is because BDQN only changes that exploration strategy of DQN, and learning the last layer which can easily accommodate additional improvements to DQN.

Game	$\frac{\text{BDQN}}{\text{DDQN}}$	$\frac{\text{BDQN}}{\text{DDQN}^{\dagger}}$	$\frac{\text{BDQN}}{\text{HUMAN}}$	Step
Amidar	558%	788%	325%	100M
Alien	103%	103%	43%	100M
Assault	396%	176%	589%	100M
Asteroids	2517%	1516%	108%	100M
Asterix	531%	385%	687%	100M
BeamRider	207%	114%	150%	70M
BattleZone	281%	253%	172%	50M
Atlantis	80604%	49413%	11172%	40M
DemonAttack	292%	114%	326%	40M
Centipede	114%	178%	61%	40M
BankHeist	211%	100%	100%	40M
CrazyClimber	148%	122%	350%	40M
ChopperCommand	14500%	1576%	732%	40M
Enduro	295%	350%	361%	30M
Pong	112%	100%	226%	5M

TABLE I

WE RUN BOTH BDQN AND DDQN FOR THE SAME NUMBER OF TIMES STEPS WRITE IN THE LAST COLUMN. THE FIRST COLUMN PRESENTS THE SCORE RATIO OF BDQN TO DDQN AFTER THE STEPS PROVIDED IN THE LAST COLUMN. THE SECOND COLUMN IS THE SCORE RATIO OF BDQN AFTER THE NUMBER OF STEPS IN THE LAST COLUMN COMPARED TO THE SCORE OF OF DDQN†, WHICH IS THE REPORTED SCORES OF DDQN IN [11] AFTER RUNNING FOR 200M SAMPLES DURING EVALUATION TIME WHERE THE $\varepsilon = 0.001$, AND THE THIRD COLUMN IS WITH RESPECT TO HUMAN SCORE REPORTED AT [8]. IT IS WORTH NOTING THAT WE DO NOT DESIGN A EVALUATION PHASE FOR BDQN

II. RELATED WORK

The complexity of the exploration-exploitation trade-off has been deeply investigated in RL literature [16], [3], [17]. [18] investigates the regret analysis of MDPs where Optimism in Face of Uncertainty (OFU) principle is deployed to guarantee a high probability regret upper bound. [19] deploys OFU in order to propose the high probability regret upper bound for Partially Observable MDPs (POMDPs) using spectral methods [20]. Furthermore, [21] tackles a general case of partial monitoring games and provides minimax regret guarantee which is polynomial in certain dimensions of the problem.

In multi-arm bandit, there are compelling empirical pieces of evidence that Thompson Sampling can provide better results than optimism-under-uncertainty approaches [22], while the state of the art performance bounds are preserved [23], [24]. A natural adaptation of this algorithm to RL, posterior sampling RL (PSRL), first proposed by [7] also shown to have good frequentist and Bayesian performance guarantees [25], [26]. Even though the theoretical RL addresses the exploration and exploitation trade-offs, these problems are still prominent in empirical reinforcement learning research [8], [27], [28]. On the empirical side, the recent success in the video games has sparked a flurry of research interest. Following the success of Deep RL on Atari games [8] and the board game Go [29], many researchers have begun exploring practical applications of deep reinforcement learning (DRL). Some investigated applications include, robotics [30], and self-driving cars [31].

Inevitably for PSRL, the act of posterior sampling for policy or value is computationally intractable with large systems, so PSRL can not be easily leveraged to high dimensional problems. To remedy these failings [32] consider the use of randomized value functions to approximate posterior samples for the value function in a computationally efficient manner. They show that with a suitable linear value function approximation, using the approximated Bayesian linear regression for randomized least-squares value iteration method can remain statistically efficient [33] but still is not scalable to large-scale RL with deep neural networks.

To combat these shortcomings, [34] suggests a bootstrapped-ensemble approach that trains several models in parallel to approximate the posterior distribution. Other works suggest using a variational approximation to the Q-networks [35] or noisy network [36]. However, most of these approaches significantly increase the computational cost of DQN and neither approach produced much beyond modest gains on Atari games. Interestingly, Bayesian approach as a technique for learning a neural network has been deployed for object recognition and image caption generation where its significant advantage has been verified [37].

In this work we present another alternative approach that extends randomized least-squares value iteration method [33] to deep neural networks: we approximate the posterior by a Bayesian linear regression only on the last layer of the neural network. This approach has several benefits, e.g. simplicity, robustness, targeted exploration, and most importantly, we find that this method is much more effective than any of these predecessors in terms of sample complexity and final performance.

Concurrently, [38] proposes least squares temporal difference which learns a linear model on the feature representation in order to estimate the Q-function while ε-greedy exploration is employed and improvement on five tested Atari games is provided. Drop-out, as another randomized exploration method is proposed by [39] but [34] investigates the sufficiency of

[1]The scores of DDQN† are borrowed from original DDQN paper [11]

the estimated uncertainty and hardness in driving a suitable exploitation out of it. As stated before, in spite of the novelties proposed by the methods, mentioned in this section, neither of them, including TS based approaches, produced much beyond modest gains on Atari games while BDQN provides significant improvements in terms of both sample complexity and final performance.

III. Thompson Sampling vs $\varepsilon-$greedy

In this section, we enumerate a few benefits of TS over ε-greedy strategies. We show how TS strategies exploit the uncertainties and expected returns to design a randomized exploration while $\varepsilon-$greedy strategies disregard all these useful information for the exploration.

In order to make a balance between exploration and exploitation, TS explores actions with higher estimated return with higher probability. In order to exploit the estimated uncertainties, TS dedicates a higher chance to explore an action if its uncertainty increases. Fig. 1*(a)* expresses the agent's estimated values and uncertainties for the available actions at a given state x. While $\varepsilon-$greedy strategy mostly focuses on the greedy action, action 1, the TS based strategy randomizes, mostly, over actions 1 through 4, utilizes their approximated expected returns and uncertainties, and with low frequency explores actions 5, 6. On the other hand, $\varepsilon-$greedy strategy explores actions 5 and 6, the actions that the RL agent is almost sure about their low expected returns, as frequent as other sub-greedy actions $2, 3, 4$ which increases its samples complexity. Moreover, a $\varepsilon-$greedy strategy requires a deep network to approximate the value of all the sub-greedy actions equally good, therefore, it dedicates the network capacity to accurately estimate the values of all the sub-greedy actions equally good, instead of focusing more on the actions with higher promising estimated value. Therefore, it ends up with not accurate enough estimation of other good actions compared to the greedy action.

In a study of value-based deep RL, e.g. DQN, the network is following a target value which is updated occasionally. Therefore, TS based strategy should not estimate the posterior distribution which adaptively follows the target values. A commonly used technique in deep RL is a moving window of replay buffer to store the recent experiences. The TS based agent, after a few tries of actions 5 and 6, builds a belief in the low return of these actions given the current target values, while it is possible that later on, the target value suggests a high expected return of these actions. Since the replay buffer is bounded moving window, lack of samples of these actions pushes the posterior belief of these actions to the prior belief, over time, and the agent tries them again in order to update its belief. Fig. 1*(b)* shows that the lack of samples for action 6 in the replay buffer, increases the uncertainty of this action and a randomized TS strategy starts to explore them over. It means that due to adaptive change of target value, respectively the objective, and limited replay buffer, the BDQN agent is never too confident about the expected return of poor actions and keeps exploring them once in a while.

In general, TS based strategy advances the exploration-exploitation balance by making a trade-off between the expected returns and the uncertainties, while $\varepsilon-$greedy strategy ignores all of this information.

Another benefit of TS over ε-greedy can be described using Fig. 1*(c)*. Consider a deterministic and episodic maze game, with episode length H of the shortest pass from the start to the destination. The agent is placed to the start point at the beginning of each episode where the goal state is to reach the destination and receive a reward of 1 otherwise the reward is 0. 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 function in the set. The agent is supposed to find a Q from this set which maximizes the average return. It is worth noting that the agent task is to find a good function from a function set.

In this situation, TS randomizes over the Q-functions with high promising returns and relatively high uncertainty, including the true Q-function. When the TS agent picks the true Q-function, it increases the posterior probability of this Q-function because it matches the observation. When the TS agent chooses other functions, they predict deterministically wrong values and the posterior update of those functions set to zero. Therefore, the agent will not choose these functions again, i.e. TS finds the true Q-function by transferring the information through posterior update which helps the agent to find the optimal Q very fast. For ε-greedy agent, even though it chooses the true function at the beginning (it is the optimistic one), at each time step, it randomizes its action with the probability ε. Therefore, it takes exponentially many trials in order to get to the target in this game.

IV. Preliminaries

An infinite horizon γ-discounted MDP M is a tuple $\langle \mathcal{X}, \mathcal{A}, T, R, \gamma \rangle$, with state space $\mathcal{X}$, action space $\mathcal{A}$, and the transition kernel T, accompanied with reward function of R where $0 < \gamma \leq 1$. 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 $\pi := \mathcal{X} \rightarrow \mathcal{A}$, a stochastic mapping from states to actions, $\pi(a|x) := \mathbb{P}(a|x)$.

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

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

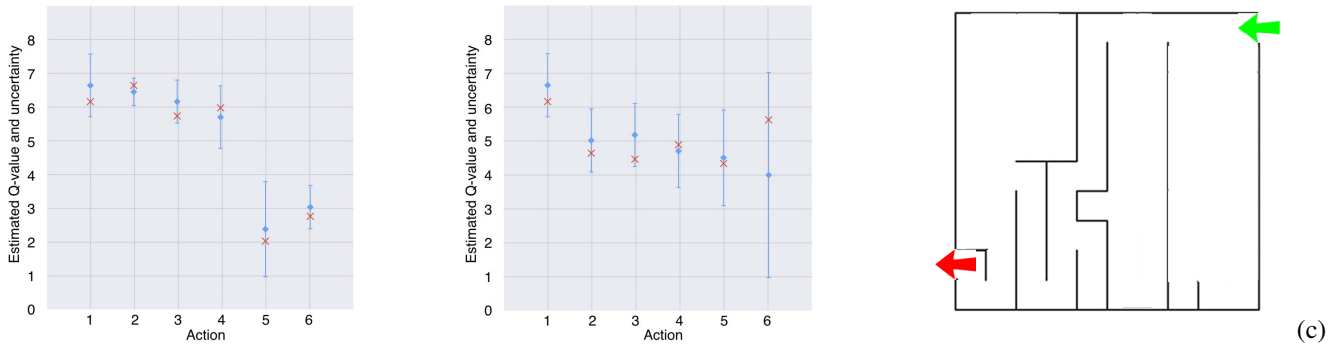

Fig. 1. A cartoon on TS vs ε-greedy. The red crosses are the target values, the diamonds are the mean of estimated Q-values with blue intervals as uncertainties (e.g. $c \cdot variance$). Description (a) ε-greedy strategy mostly chooses the greedy action, action 1 and explore actions $5, 6$ as much as actions $2, 3, 4$, while TS randomizes mostly over actions $1, 2, 3, 4$ barely explore actions $5, 6$. (b) BDQN computes the posterior using recent experiences in the replay buffer. Therefore, lack of samples of action 6 increases the uncertainty on the estimated value and BDQN explores it again. This further exploration is crucial since the target value changes over time. (c) maze.

starting off from state x and taking action a in the first place.

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

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

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

To find the optimal policy, one can solve a linear programming problem in Eq. 1 or follow the corresponding Bellman equation Eq. 2 where both of the optimization methods solve the following

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

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

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

has been proposed [40], [41]. Here, the tuple (x, a, r, x', a') consists of consecutive samples under behavioral policy π. Furthermore, [8] carries the same idea, and introduce Deep Q-Network (DQN) where the Q-functions are parameterized by a deep network. To improve the quality of Q estimate, they use back propagation on loss $\mathcal{L}(Q)$ using the TD update [42]. In the following we describe the setting used in DDQN. In order to reduce the bias of the estimator, they introduce target network Q^{target} and target value $y = r + \gamma Q^{target}(x', \hat{a})$ where $\hat{a} = \arg\max_{a'} Q(x', a')$ with a new loss $\mathcal{L}(Q, Q^{target})$

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

This regression problem minimizes the estimated loss $\widehat{\mathcal{L}}(Q, Q^{target})$, which minimize the distance between the Q and the target y. A DDQN agent, once in a while updates the Q^{target} network by setting it to Q network, pursues the regression with the new target value and provides a biased estimator of the target.

V. Bayesian Deep Q-Networks

We propose a Bayesian method to approximate the Q-function and match it to the target value. We utilize the DQN architecture, remove its last layer, and directly build a Bayesian linear regression (BLR) [43] on the output of the deep network $\phi_\theta(x) \in \mathbb{R}^d$, the feature representation layer, parametrized by θ. We deploy BLR to efficiently approximate the distribution over the Q-values where the uncertainty over the values is captured. A common assumption in DNN is that the feature representation is suitable for linear classification or regression (same assumption in DQN), therefore, building a linear model on the feature a suitable choice.

The Q-functions can be approximated as a linear transformation of features, i.e. for a given pair of state-action, $Q(x,a) = \phi_\theta(x)^\top w_a$, where $w_a \in \mathcal{R}^d, \forall a \in \mathcal{A}$. Consequently, as mentioned in the previous section, the target value is generated using target model. The target model follows the same structure as the Q model, and contains $\phi_\theta^{target}(x) \in \mathbb{R}^d, \forall x \in \mathcal{X}$ denotes the feature representation of target network, and $w^{target}{}_{\hat{a}}$, $\forall \hat{a} \in \mathcal{A}$ denotes the target linear model applied on the target feature representation. Inspired by DDQN, for a given tuple of experience (x, a, r, x'), the predicted value of pair (x, a) is $Q(x,a) = \phi_\theta(x)^\top w_a$, while the target value is

$$y = r + \gamma \phi^{target}(x') w^{target}{}_{\hat{a}}, \ ; \hat{a} = argmax_a \phi_\theta(x')^\top w_a.$$

Therefore, by deploying BLR on the space of features, we

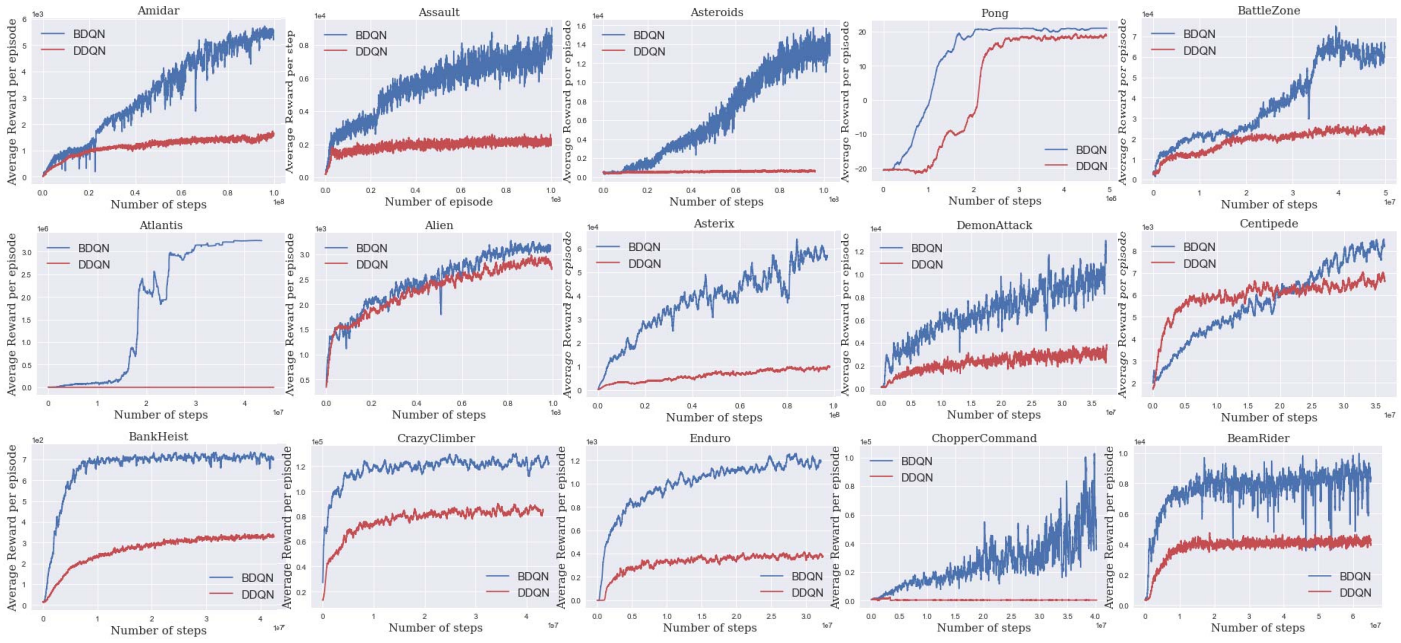

Fig. 2. The efficient and targeted exploration of BDQN

can approximate the posterior distribution of model parameter $w_a \in \mathcal{R}^d, \forall a \in \mathcal{A}$, as well as the posterior distribution of the Q-functions using the corresponding target values. In Gaussian BLR models, in order to make the posterior update computationally tractable in a closed form a common approximation is to make the prior and likelihood choices as conjugates of each other. Therefore, for a given pair of (x, a), the vector w_a is drawn from a Gaussian prior $\mathcal{N}(0, \sigma^2)$ and given w_a, the target value is generated from the following model;

$$y \sim w_a^\top \phi(x) + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is an iid noise. 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 $\mathbb{P}(y|\, a) = \int_{w_a} \mathbb{P}(y|w_a)\, \mathbb{P}(w_a)\, dw_a$ which also has a closed form.

Given a experience replay buffer $\mathcal{D} = \{x_\tau, a_\tau, y_\tau\}_{\tau=1}^D$, we construct $|\mathcal{A}|$ (number of actions) disjoint datasets for each action, $\mathcal{D}_a$, where $\mathcal{D} = \cup_{a\in\mathcal{A}}\mathcal{D}_a$ and $\mathcal{D}_a$ is a set of tuples x_τ, a_τ, y_τ with the action $a_\tau = a$ and cardinality D_a. We are interested in the approximated posterior distribution of w_a, $\forall a \in \mathcal{A}$ and correspondingly the $Q(x, a)$; $\mathbb{P}(w_a|\mathcal{D}_a)$, $\forall a \in \mathcal{A}$ and $\mathbb{P}(Q(x, a)|\mathcal{D}_a), \forall x \in \mathcal{X}$.

Following steps are the standard BLR steps and we suggest for readers who are confident in BLR to skip the derivation. For each action a and the corresponding dataset D_a, we 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 of w_a is as follows:

$$w_a \sim \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2}\Xi_a\Phi_a\mathbf{y}_a, \Xi_a\right) \tag{5}$$

where

$$\Xi_a = \left(\frac{1}{\sigma_\epsilon^2}\Phi_a\Phi_a^\top + \frac{1}{\sigma^2}\mathbb{I}\right)^{-1}$$

and $\mathbb{I} \in \mathbb{R}^d$ is an identity matrix. The $Q(x, a)|\mathcal{D}_a = w_a^\top\phi(x)$ where w_a is drawn following the posterior distribution in Eq. 5. Since the prior and likelihood are conjugate of each other we have the closed form posterior distribution of the discounted return approximated as

$$\sum_{t=0}^{N}\gamma^t r_t|x_0 = x, a_0 = a, \mathcal{D}_a \sim \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2}\phi(x)^\top\Xi_a\Phi_a\mathbf{y}_a, \phi(x)^\top\Xi_a\phi(x)\right) \tag{6}$$

As TS suggests, for the exploration, we exploit the expression in Eq. 5.At the decision time, we sample a wight vector w_a for each action in order to have samples of Q-values. Then we act optimally with respect to these sampled value

$$a_{\text{TS}} = \arg\max_a w_a^\top\phi_\theta(x). \tag{7}$$

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 experience replay buffer where the agent stores its recent experiences. The agent draws $W \sim \mathcal{N}(W^{target}, Cov)$ (abbreviation for sampling of vector w_a for each action separately) every T^{sample} steps and act optimally with respect to the drawn weights. During the inner loop of the algorithm, we draw a minibatch of data from replay buffer

and use loss

$$(y_\tau - \left[W^\top \phi_\theta(x_\tau)\right]_{a_\tau})^2 \tag{8}$$

$$\textit{where} \quad y_\tau := r_\tau + \left[W^{target^\top} \phi_{\theta^{target}}(x_{\tau+1})\right]_{\hat{a}}$$
$$\hat{a} := argmax_{a'} \left[W^\top \phi_\theta(x_{\tau+1})\right]_{a'} \tag{9}$$

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

We update the target network every T^{target} steps and set θ^{target} to θ. With the period of $T^{Bayes\ target}$ the agent updates its posterior distribution using a larger minibatch of data drawn from replay buffer, set the w_a^{target}, $\forall a \in \mathcal{A}$ to the mean of the posterior, and sample w_a, $\forall \mathcal{A}$ with respect to the updated posterior. Algorithm 1 gives the full description of BDQN.

VI. Experiments

We apply BDQN on a variety of Atari games using the Arcade Learning Environment [9] through OpenAI Gym[2] [44]. As a baseline[3], we run the DDQN algorithm and evaluate BDQN on the measures of sample complexity and score.

a) Network architecture: : The input to the network part of BDQN is $4\times84\times84$ tensor with a rescaled and averaged over channels 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 layer with size 512. We add a BLR layer on top of this.

b) Choice of hyper-parameters: : For BDQN, we set the values of W^{target} to the mean of the posterior distribution over the weights of BLR with covariances Cov and draw W from this posterior. For the fixed W and W^{target}, we randomly initialize the parameters of network part of BDQN, θ, and train it using RMSProp, with learning rate of 0.0025, and a momentum of 0.95, inspired by [8] where the discount factor is $\gamma = 0.99$, the number of steps between target updates $T^{target} = 10k$ steps, and weights W are re-sampled from their posterior distribution every T^{sample} steps. We update the network part of BDQN every 4 steps by uniformly at random 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 (if the size of replay buffer is less than B at the current step, we choose the minimum of these two), with entries sampled uniformly form replay buffer. The experience replay contains the $1M$ most recent transitions. Further hyper-parameters are equivalent to ones in DQN setting.

[2]Each input frame is a pixel-max of the two consecutive frames. We detailed the environment setting in the implementation code

[3]We also attempted to include Bootstrapped DQNs [34] as a baseline. Due to the lack of implementation details, we tried different reasonable way to make it work but we were not successful to reproduce it despite extensive experimentation. Moreover, they admit in their paper that uniformly chosen DQNs outperform their setting. Our bootstrap DQN implementation is publicly available

Algorithm 1 BDQN

1: Initialize parameter sets θ, θ^{target}, W, W^{target}, and Cov using a normal distribution.
2: Initialize replay buffer and set counter $= 0$
3: **for** episode = 1 to $\inf$ **do**
4: Initialize x_1 to the initial state of the environment
5: **for** $t =$ to the end of episode **do**
6: **if** count mod $T^{sample} = 0$ **then**
7: sample $W \sim \mathcal{N}(W^{target}, Cov)$
8: **end if**
9: Select action $a_t = argmax_{a'} \left[W^\top \phi_\theta(x_t)\right]_{a'}$
10: Execute action a_t in environment, observing reward r_t and successor state x_{t+1}
11: Store transition (x_t, a_t, r_t, x_{t+1}) in replay buffer
12: Sample a random minibatch of transitions (x_τ , a_τ, r_τ, $x_{\tau+1}$) from replay buffer
13: $y_\tau \leftarrow \begin{cases} \text{for terminal } x_{\tau+1}: \\ \quad r_\tau \\ \text{for non-terminal } x_{\tau+1}: \\ \quad r_\tau + \left[W^{target^\top} \phi_{\theta^{target}}(x_{\tau+1})\right]_{\hat{a}} \textit{ where} \\ \quad \hat{a} = argmax_{a'} \left[W^\top \phi_\theta(x_{\tau+1})\right]_{a'} \end{cases}$
14: $\theta \leftarrow \theta - \eta \cdot \nabla_\theta (y_\tau - \left[W^\top \phi_\theta(x_\tau)\right]_{a_\tau})^2$
15: **if** count mod $T^{target} = 0$ **then**
16: set $\theta^{target} \leftarrow \theta$
17: **end if**
18: **if** count mod $T^{Bayes\ target} = 0$ **then**
19: Update W^{target} and Cov
20: **end if**
21: count = count + 1
22: **end for**
23: **end for**

c) Hyper-parameters tuning: : 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 optimize for this set of hyper-parameters we set up a very simple, fast, and cheap hyper-parameter tunning procedure which proves the robustness of BDQN. To fine the first three, we set up a simple hyper-parameter search. We used a pretrained DQN model for the game of *Assault*, and removed the last fully connected layer in order to have access to its already trained 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 hyper-parameter tuning is cheap and fast since it requires only a few times the B number of forwarding 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 choose the

Game	BDQN	DDQN	DDQN†	Human	SC	SC†	Step
Amidar	5.52k	0.99k	0.7k	1.7k	22.9M	4.4M	100M
Alien	3k	2.9k	2.9k	6.9k	-	36.27M	100M
Assault	8.84k	2.23k	5.02k	1.5k	1.6M	24.3M	100M
Asteroids	14.1k	0.56k	0.93k	13.1k	58.2M	9.7M	100M
Asterix	58.4k	11k	15.15k	8.5k	3.6M	5.7M	100M
BeamRider	8.7k	4.2k	7.6k	5.8k	4.0M	8.1M	70M
BattleZone	65.2k	23.2k	24.7k	38k	25.1M	14.9M	50M
Atlantis	3.24M	39.7k	64.76k	29.0k	3.3M	5.1M	40M
DemonAttack	11.1k	3.8k	9.7k	3.4k	2.0M	19.9M	40M
Centipede	7.3k	6.4k	4.1k	12.0k	-	4.2M	40M
BankHeist	0.72k	0.34k	0.72k	0.72k	2.1M	10.1M	40M
CrazyClimber	124k	84k	102k	35.4k	0.12M	2.1M	40M
ChopperCommand	72.5k	0.5k	4.6 k	9.9k	4.4M	2.2M	40M
Enduro	1.12k	0.38k	0.32k	0.31	0.82M	0.8M	30M
Pong	21	18.82	21	9.3	1.2M	2.4M	5M

TABLE II

COMPARISON OF SCORES AND SAMPLE COMPLEXITIES(SCORES IN THE FIRST TWO COLUMNS ARE AVERAGE OF 100 CONSECUTIVE EPISODES). THE SAMPLE COMPLEXITY, *SC*, REPRESENTS THE NUMBER OF SAMPLES THE BDQN REQUIRES TO BIT THE HUMAN SCORE [8](" $-$ " MEANS BDQN COULD NOT BIT) AND *SC*† IS THE NUMBER NUMBER OF SAMPLES THE BDQN REQUIRES TO BIT THE SCORE OF DDQN †.

higher one. We started off with the learning rate of 0.0025 and did not tune for that. Thanks to the efficient TS exploration and closed form BLR, BDQN can learn a better policy in an even shorter period of time. In contrast, it is well known for DQN based methods that changing the learning rate causes a major degradation in the performance, Apx. VIII. The proposed hyper-parameter search is very simple where the exhaustive hyper-parameter search is likely to provide even better performance. In order to compare the fairness in sample usage, we argue in Apx. VIII, that the network part of BDQN and its corresponding part in DDQN observe the same number of samples but the BLR part of BDQN uses 16 times fewer samples compared to its corresponding last layer in DDQN, Apx. VIII.

All the implementations are coded in MXNet framework [45] and are available at ... (The code, trained models, and recorded arrays of returns are all publicly available and in order to compare against BDQN, there is not need to run the usually massive and expensive deep RL experiments. Due to the double-blind review process, the link can not be provided).

d) Results: : The results are provided in Fig. 2 and Table. II. Mostly the focus of the experiments are on sample complexity in Deep-RL, even though, BDQN provides much larger scores compared to base line. For example, for the game *Atlantis*, DDQN† gives score of $64.67k$ after $200M$ samples during evaluation time, while BDQN reaches $3.24M$ after $40M$ samples. As it is been shown in Fig. 2, BDQN saturates for *Atlantis* after 20M samples. We realized that BDQN reaches the internal *OpenAIGym* limit of $max\_episode$, where relaxing it improves score after $15M$ steps to $62M$.

We can observe that BDQN can immediately learn significantly better policies due to its targeted exploration in a much shorter period of time. Since BDQN on game *Atlantis* promise a big jump around time step $20M$, we ran it five more times in order to make sure it was not just a coincidence. We did the same additional five experiments for the game *Amidar* as well. We observed that the improvements are consistent among the different runs. For the game Pong, we ran the experiment for a longer period but just plotted the beginning of it in order to observe the difference. For some games, we did not run the experiment to $100M$ samples since the reached their plateau.

VII. CONCLUSION

In this work we proposed BDQN, a practical TS based RL algorithm which provides targeted exploration in a computationally efficient manner. It involved making simple modifications to the DDQN architecture by replacing the last layer with a Bayesian linear regression. Under the Gaussian prior, we obtained fast closed-form updates for the posterior distribution. We demonstrated significantly faster training and enormously better rewards over a strong baseline DDQN.

This suggests that BDQN can benefit even more from further modifications to DQN such as e.g. Prioritized Experience Replay [12], Dueling approach [13], A3C [14], safe exploration [15], and etc. We plan to explore the benefits of BDQN formulation in the future works.

Moreover, concurrent to BDQN, a similar approach is proposed in NoisyNet [36] where the randomization over all the weights of the network is deployed. This approach, after architecture engineering, at least doubles the computation cost of DDQN but improves the overall performance in Atari

games. Since BDQN just randomizes the last layer of the model and outperforms NoisyNet in 13 out of 15 we plan to study NoisyNet where the randomization happens just in the last layer. This approach does not increase the computation cost of DDQN, since the last layer has a negligible fraction of the model parameters and it is a nice and direct extension of TS of contextual linear bandits [46], [47].

In RL, policy gradient [48], [49], [50] is another approach which directly learn the policy. In practical policy gradient, even though the optimal policy, given Markovian assumption, needs to be deterministic, the policy regularization is a dominant approach to make the policy stochastic and preserve the exploration thorough the stochasticity of the policy [51], [52]. We plan to explore the advantage of TS based exploration instead of regularizing the policy and make it stochastic.

VIII. Appendix

a) Learning rate:: It is well known that DQN and DDQN are sensitive to the learning rate and change of learning rate can degrade the performance to even worse than random policy. We tried the same learning rate as BDQN, 0.0025, for DDQN and observed that its performance drops. Fig. 3 shows that the DDQN with higher learning rates learns as good as BDQN at the very beginning but it can not maintain the rate of improvement and degrade even worse than the original DDQN.

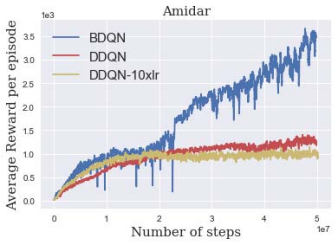

Fig. 3. Effect of learning rate on DDQN

b) Computational and sample cost comparison: : For a given period of game time, the number of the 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, sees the same number of samples, but the last layer of BDQN sees 16 times fewer samples compared to the last layer of DQN. The last layer of DQN for a duration of $100k$, observes $25k = 100k/4$ (4 is back prob period) mini batches 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. Therefore, during the duration of $100k$ decision making steps, for the learning procedure, DDQN does $32 * 25k$ of forward passes and $32 * 25k$ of backward passes, while BDQN does same number of backward passes (cheaper since there is no backward pass for the final layer) and $36 * 25k$ of forward passes. One can easily relax it by parallelizing this step along the main body of BDQN or deploying on-line posterior update methods.

c) Thompson sampling frequency:: The choice of TS update frequency can be crucial from domain to domain. If one chooses T^{sample} too short, then computed gradient for backpropagation of the feature representation is not going to be useful since the gradient get noisier and the loss function is changing too frequently. On the other hand, the network tries to find a feature representation which is suitable for a wide range of different weights of the last layer, results in improper use of model capacity. If the TS update frequency is too low, then it is far from being TS and losses randomized exploration property. The current choice of T^{sample} is suitable for a variety of Atari games since the length of each episode is in range of $\mathcal{O}(T^{sample})$ and is infrequent enough to make the feature representation robust to big changes.

For the RL problems with shorter horizon we suggest to introduce two more parameters, $\tilde{T}^{sample}$ and $\tilde{W}$ where $\tilde{T}^{sample}$, the period that of $\tilde{W}$ is sampled our of posterior, is much smaller than T^{sample} and $\tilde{W}$ is being used just for making TS actions while W is used for backpropagation of feature representation. For game Assault, we tried using $\tilde{T}^{sample}$ and $\tilde{W}$ but did not observe much a difference, and set them to T^{sample} and W. But for RL setting with a shorter horizon, we suggest using them.

d) Further investigation in Atlantis:: After removing the maximum episode length limit for the game Atlantis, BDQN gets the score of 62M. This episode is long enough to fill half of the replay buffer and make the model perfect for the later part of the game but losing the crafted skill for the beginning of the game. We observe in Fig. 4 that after losing the game in a long episode, the agent forgets a bit of its skill and loses few games but wraps up immediately and gets to score of $30M$. To overcome this issue, one can expand the replay buffer size, stochastically store samples in the reply buffer where the later samples get stored with lowers chance, or train new models for the later parts of the episode. There are many possible cures for this interesting observation and while we are comparing against DDQN, we do not want to advance BDQN structure-wise.

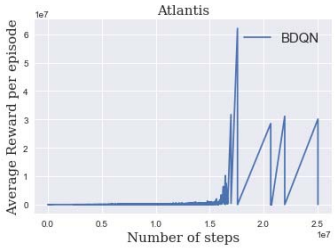

Fig. 4. BDQN on Atlantis after removing the limit on max of episode length

Acknowledgment

The authors would like to thank Zachary C. Lipton, Marlos C. Machado, Ian Osband, Gergely Neu, and the anonymous reviewers for their feedback and suggestions.

REFERENCES

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

[2] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Exploration:a study of count-based exploration for deep reinforcement learning," *arXiv*, 2016.

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

[4] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, 1933.

[5] A. N. Sanborn and N. Chater, "Bayesian brains without probabilities," *Trends in cognitive sciences*, vol. 20, no. 12, pp. 883–893, 2016.

[6] M. Ghavamzadeh, S. Mannor, J. Pineau, A. Tamar *et al.*, "Bayesian reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, 2015.

[7] M. Strens, "A bayesian framework for reinforcement learning," in *ICML*, 2000.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents." *J. Artif. Intell. Res.(JAIR)*, 2013.

[10] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *arXiv preprint arXiv:1709.06009*, 2017.

[11] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, 2016.

[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv*, 2015.

[13] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv*, 2015.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016.

[15] Z. C. Lipton, J. Gao, L. Li, J. Chen, and L. Deng, "Combating reinforcement learning's sisyphean curse with intrinsic fear," *arXiv preprint arXiv:1611.01211*, 2016.

[16] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, vol. 49, no. 2-3, pp. 209–232, 2002.

[17] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate, "A bayesian sampling approach to exploration in reinforcement learning," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009.

[18] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *Journal of Machine Learning Research*, 2010.

[19] K. Azizzadenesheli, A. Lazaric, and A. Anandkumar, "Reinforcement learning of pomdps using spectral methods," in *Proceedings of the 29th Annual Conference on Learning Theory (COLT)*, 2016.

[20] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 2773–2832, 2014.

[21] G. Bartók, D. P. Foster, D. Pál, A. Rakhlin, and C. Szepesvári, "Partial monitoringclassification, regret bounds, and algorithms," *Mathematics of Operations Research*, 2014.

[22] O. Chapelle and L. Li, "An empirical evaluation of thompson sampling," in *Advances in neural information processing systems*, 2011, pp. 2249–2257.

[23] D. Russo and B. Van Roy, "Learning to optimize via posterior sampling," *Mathematics of Operations Research*, vol. 39, no. 4, pp. 1221–1243, 2014.

[24] S. Agrawal and N. Goyal, "Analysis of thompson sampling for the multi-armed bandit problem." in *COLT*, 2012.

[25] I. Osband, D. Russo, and B. Van Roy, "(more) efficient reinforcement learning via posterior sampling," in *Advances in Neural Information Processing Systems*, 2013.

[26] Y. Abbasi-Yadkori and C. Szepesvári, "Bayesian optimal control of smoothly parameterized systems." in *UAI*, 2015, pp. 1–11.

[27] D. Abel, A. Agarwal, F. Diaz, A. Krishnamurthy, and R. E. Schapire, "Exploratory gradient boosting for reinforcement learning in complex domains," *arXiv*, 2016.

[28] K. Azizzadenesheli, A. Lazaric, and A. Anandkumar, "Reinforcement learning in rich-observation mdps using spectral methods," *arXiv preprint arXiv:1611.03907*, 2016.

[29] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, 2017.

[30] S. Levine et al., "End-to-end training of deep visuomotor policies," *JMLR*, 2016.

[31] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv*, 2016.

[32] I. Osband, D. Russo, Z. Wen, and B. Van Roy, "Deep exploration via randomized value functions," *arXiv*, 2017.

[33] I. Osband, B. Van Roy, and Z. Wen, "Generalization and exploration via randomized value functions," *arXiv*, 2014.

[34] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *Advances in Neural Information Processing Systems*, 2016.

[35] Z. C. Lipton, J. Gao, L. Li, X. Li, F. Ahmed, and L. Deng, "Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking," *arXiv preprint arXiv:1608.05081*, 2016.

[36] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[37] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable bayesian optimization using deep neural networks," in *ICML*, 2015.

[38] N. Levine, T. Zahavy, D. J. Mankowitz, A. Tamar, and S. Mannor, "Shallow updates for deep reinforcement learning," *arXiv preprint arXiv:1705.07461*, 2017.

[39] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016.

[40] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of machine learning research*, vol. 4, no. Dec, pp. 1107–1149, 2003.

[41] A. Antos, C. Szepesvári, and R. Munos, "Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path," *Machine Learning*, 2008.

[42] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[43] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.

[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[45] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv*, 2015.

[46] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *ICML*, 2013, pp. 127–135.

[47] M. Abeille and A. Lazaric, "Linear thompson sampling revisited," in *AISTATS 2017-20th International Conference on Artificial Intelligence and Statistics*, 2017.

[48] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000.

[49] S. M. Kakade, "A natural policy gradient," in *Advances in neural information processing systems*, 2002.

[50] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015.

[51] G. Neu, A. Jonsson, and V. Gómez, "A unified view of entropy-regularized markov decision processes," *arXiv*, 2017.

[52] J. Schulman, P. Abbeel, and X. Chen, "Equivalence between policy gradients and soft q-learning," *arXiv*, 2017.