Uncertainty Based Exploration in Reinforcement Learning

Analyzing the Robustness of Bayesian Deep Q-Networks

Sagi Schwartz[1]
Supervisors: Neil Yorke-Smith[1]**, Pascal van der Vaart**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
July 6, 2025

Name of the student: Sagi Schwartz
Final project course: CSE3000 Research Project
Thesis committee: Neil Yorke-Smith, Pascal van der Vaart, Matthijs Spaan

Abstract

Bayesian Deep Q-Networks (BDQN) have demonstrated superior exploration capabilities and performance in complex environments such as Atari games, yet their behavior in other simpler settings and their sensitivity to hyperparameters remain understudied. This work evaluates BDQN in both contextual bandit and reinforcement learning tasks, compares it against the standard ϵ-greedy exploration strategy and analyzes its hyperparameter sensitivity. Our results indicate that BDQN outperforms ϵ-greedy DQN in exploration-heavy environments, particularly Deep Sea with sparse rewards, but performs comparably in simpler tasks where exploration is less critical. Sensitivity analysis reveals that the forgetting factor (α) plays a central role in modulating exploration, while other hyperparameters such as batch size also impact performance to varying degrees. These findings suggest BDQN is a promising strategy for complex tasks requiring persistent exploration, though it introduces additional tuning complexity.

1 Introduction

Reinforcement learning (RL) is a subfield of machine learning focused on learning optimal policies that maximize rewards through agent-environment interactions. Although RL approaches increasingly outperform humans in environments such as Atari games, they suffer from a major limitation of poor sample efficiency. One of the sources of this problem is naive exploration strategies, such as random exploration [16], which can lead to billions of steps taken by an agent before reaching human-level capabilities. With random exploration, the agent picks its next action randomly regardless of prior experience. This type of exploration is incorporated into the ϵ-greedy strategy, which reduces exploration as more experience with the environment is gained, using the decaying parameter ϵ. Due to its lack of efficiency and simple exploration approach, ϵ-greedy often struggles in more complex environments.

Multiple exploration methods have been suggested as a remedy in recent years. One of the approaches that has been shown to be successful in practice is the Bayesian deep Q-networks (BDQN) [2]. This approach relies on double DQN (DDQN) [7] where the last linear regression layer is replaced by Bayesian linear regression (BLR). The algorithm then applies Thompson sampling on the approximated posteriors to balance the exploration-exploitation trade-off.

Azizzadenesheli and Anandkumar [2] compared the performance of BDQN in Atari 2600 games to several other algorithms such as DDQN, Bootstrap DQN [14] and NoisyNet [6], which BDQN outperformed in most games. However, the success of the algorithm in Atari games does not give a full picture of the algorithm's performance and robustness in different settings and other environments. Specifically, existing literature does not examine the algorithm's performance on simpler environments, such as contextual multi-armed bandits, nor does it investigate its sensitivity to its hyperparameters.

This is interesting because, unlike high-dimensional environments such as Atari games, contextual bandits provide a controlled setting that allows for a more precise evaluation of the algorithm's behavior and generalization properties. Understanding BDQN's performance in these environments helps to determine whether its advantages stem from its core innovations or from specific characteristics of more complex domains. Moreover, contextual bandits are widely used in real-world applications (e.g. recommender systems, clinical trial) [4] and evaluating BDQN in these settings has practical relevance. Furthermore, hyperparameter sensitivity is a known understudied challenge in deep RL. Analyzing BDQN sensitivity to its hyperparameters is crucial to understanding its stability and the impact of small changes in hyperparameters on its overall performance. A useful framework for doing so was suggested by Adkins, Bowling, and White [1] and is explained in 2.6.

This paper aims to fill the described knowledge gap by answering the following questions:

1. Performance - How does BDQN's exploration method perform in solving RL environments, compared to the ϵ-greedy exploration strategy?
2. Sensitivity - To which BLR hyperparameters is BDQN sensitive? What is the effect of each BLR hyperparameter?
3. Transferability - Does BDQN performance transfer well across tasks? Do optimal hyperparameter values remain stable or vary?

To address these questions, we implement and evaluate the BDQN algorithm in multiple benchmark RL environments. We systematically compare its performance with the ϵ-greedy exploration, perform a detailed sensitivity analysis of its Bayesian linear regression (BLR) hyperparameters, and evaluate its generalization across tasks. Our findings offer insights on the effectiveness and robustness of BDQN's exploration strategy for deep RL.

2 Background

This work analyzes the performance of BDQN's exploration-exploitation strategy under different conditions. This section explains the inner workings of the algorithm and the concepts related to it, the metrics used to measure hyperparameter sensitivity, and the environments used in our research.

2.1 Markov Decision Processes (MDPs)

RL problems are typically formalized using the framework of Markov Decision Processes (MDPs). An MDP is a mathematical framework for modeling decision-making in stochastic environments. It is defined by the tuple (S, A, P_a, R_a), where S is the state space, A is the action space, $P_a(s, s')$ is the transition probability of moving from state s to s' after taking action a, and $R_a(s, s')$ is the immediate reward for that transition. A policy π maps each state to an action and its goal is to find a policy that maximizes the expected cumulative reward

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})\right]$$

where actions are chosen according to the policy, $\gamma \in [0, 1]$ is a discount factor, and t is the time step in the environment.

2.2 Deep Q-Networks (DQN)

Before introducing any type of deep learning into RL, it is essential to cover Q-learning [19], a fundamental approach to learning a policy whose principles are used in many of the state-of-the-art algorithms. Q-learning is an off-policy algorithm which estimates the optimal action values by updating

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[R_{a_t}(s_t, s_{t+1}) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\Big]$$

where $Q(s_t, a_t)$ is the estimated Q-value of taking action a_t in state s_t and α is the learning rate. However, learning sequentially from consecutive agent experiences in the environment could be unstable and inefficient. As a remedy, experience replay, introduced by Lin [10] is often used. With experience replay, instead of learning from consecutive experiences, which can be highly correlated and lead to inefficient updates, the agent stores its past experiences - each consisting of the current state, action taken, reward received, next state, and whether the episode ended - in a replay buffer. During training, the agent randomly samples mini-batches of experiences from this buffer to update its policy or value function. This random sampling breaks the temporal correlations between experiences and allows the algorithm to reuse valuable past data multiple times, improving sample efficiency and overall performance.

Mnih et al. [11] coupled Q-learning and experience replay with a convolutional deep neural network to learn control policies from raw pixel input. In the paper, the authors refer to the resulting algorithm as Deep Q-Networks (DQN).

Despite its advances, the Q-learning algorithm introduced by Watkins [19] and employed by Mnih et al. [11] suffers from an overly optimistic approximation of Q-values. This overestimation can adversly affect performance, as demonstrated by Hasselt, Guez, and Silver [7], who proposed the Double DQN (DDQN) algorithm to mitigate this issue. DDQN reduces overestimation bias by decoupling action selection and action evaluation. Specifically, the *online network* (which is continuously updated) is used to select the action with the highest estimated Q-value, while the *target network* (a more stable copy of the online network) is used to evaluate it. The target value is calculated as $y = R_{a_t}(s_t, s_{t+1}) + \gamma Q_{target}(s_{t+1}, \arg\max_{a'} Q_{online}(s_{t+1}, a'))$, and the algorithm updates the online network by minimizing the loss $\mathcal{L}(Q_{online}, Q_{\text{target}}) = \mathbb{E}\left[(Q_{online}(s_t, a_t) - y)^2\right]$. Periodically, the target network is synchronized with the online network to maintain stability.

2.3 Bayesian Linear Regression

Bayesian linear regression is a probabilistic approach to modeling the relationship between a set of input variables and a target variable. Unlike classical linear regression, which estimates a single set of parameters by minimizing a loss function, the Bayesian framework treats the model parameters as random variables with prior distributions. Given a linear model $y = X\beta + \epsilon$, where noise $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, we impose a prior distribution on the coefficients β, typically a multivariate normal distribution $\mathcal{N}(0, \tau^2 I)$. In our case, σ^2 is the noise variance, τ^2 is the prior's variance, and I is the identity matrix. Upon observing data, Bayes' theorem is used to update the prior into a posterior distribution over the parameters, combining prior beliefs with the likelihood of the observed data.

The posterior distribution for the parameters β remains Gaussian due to the conjugacy of the normal prior and likelihood. Specifically, the posterior mean and covariance can be derived analytically, leading to closed-form expressions for inference. This posterior not only provides point estimates, but also quantifies uncertainty in the predictions. This makes Bayesian linear regression valuable in applications where understanding the confidence in a prediction is crucial. The exact mathematical formulation relevant to this paper is given in Section 2.5.

2.4 Thompson Sampling

Thomson Sampling [17, 18] is a heuristic that manages the exploration-exploitation trade-off in problems such as multi-armed bandit. The key idea is to maintain a probability distribution (usually Bayesian) over the parameters of each possible action's reward. At each step, the algorithm samples a value from each action's distribution and selects the action with the highest sampled value. This naturally balances exploration and exploitation, since actions with uncertain but potentially high rewards are more likely to be sampled early on. Over time, as more data are collected, the distributions become more accurate and the algorithm increasingly favors the best-performing actions.

2.5 Bayesian Deep Q-Network (BDQN)

Bayesian deep Q-networks (BDQN) which was purposed by Azizzadenesheli and Anandkumar [2] is an extension of Double DQN (DDQN) [7] that incorporates efficient, uncertainty-driven exploration using Bayesian linear regression (BLR) and Thompson sampling. The key innovation is to introduce Bayesian modeling only at the last layer of the Q-network, which allows for efficient posterior updates and exploration, while keeping the rest of the network as in standard DDQN.

As described in the original paper [2], let $\phi(s,a)$ denote the feature representation from the penultimate layer of the neural network. The Q-values are modeled as $Q(s,a) = \phi(s,a)^\top w_a$ where w is the weight vector for the output layer. In BDQN, w is modeled using Guassian BLR instead of plain linear regression, resulting in an approximate posterior distribution for w and for $Q(x,a)$. This last layer provides uncertainty over the Q-value estimates and is coupled with Thompson sampling for an efficient exploration-exploitation method.

Given an experience replay buffer $\mathcal{D} = \{s_\tau, a_\tau, y_\tau\}_{\tau=1}^{D}$, we isolate a dataset $\mathcal{D}_a$ for each action a, where $a_\tau = a$. From this subset, we form the matrix $\Phi_a^\theta \in \mathbb{R}^{d\times|\mathcal{D}_a|}$, which is the concatenation of the feature vectors $\{\phi_\theta(s_i)\}_{i=1}^{|\mathcal{D}_a|}$, and the vector $\mathbf{y}_a \in \mathbb{R}^{|\mathcal{D}_a|}$, which stacks the corresponding target values.

We estimate the posterior distribution of the weight vector w_a using the following equations:

$$\overline{w}_a := \frac{1}{\sigma_\epsilon^2}\mathrm{Cov}_a \Phi_a^\theta \mathbf{y}_a \tag{1}$$

$$\mathrm{Cov}_a := \left(\frac{1}{\sigma_\epsilon^2}\Phi_a^\theta {\Phi_a^\theta}^\top + \frac{1}{\sigma^2} I\right)^{-1} \tag{2}$$

which results in posterior samples $w_a \sim \mathcal{N}(\overline{w}_a, \mathrm{Cov}_a)$. This formulation corresponds to the standard Bayesian linear regression (BLR) setup, assuming a zero-mean prior and standard deviations of σ and σ_ϵ for the prior and likelihood, respectively.

Furthermore, the implementation of BDQN accumulates $\Phi_a^\theta {\Phi_a^\theta}^\top$ and $\Phi_a^\theta \mathbf{y}_a$ every update of the posterior. To that end, a forgetting factor α (a constant between 0 and 1) is introduced as a coefficient for both of these terms, such that at each update t, $\Phi_a^\theta {\Phi_a^\theta}^\top(t) \leftarrow (1-\alpha)\cdot \Phi_a^\theta {\Phi_a^\theta}^\top(t-1) + \Phi_a^\theta {\Phi_a^\theta}^\top(t)$ and $\Phi_a^\theta \mathbf{y}_a(t) \leftarrow (1-\alpha)\cdot \Phi_a^\theta \mathbf{y}_a(t-1) + \Phi_a^\theta \mathbf{y}_a(t)$. This reduces the impact of previously sampled data and gives more importance to recently sampled data.

Lastly, since $Q(x,a) \mid \mathcal{D}_a$ is a linear transformation of Gaussian w_a, we have:

$$Q(s,a) \mid \mathcal{D}_a \sim \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2}\phi(s)^\top \mathrm{Cov}_a \Phi_a \mathbf{y}_a,\ \phi(s)^\top \mathrm{Cov}_a \phi(s)\right) \tag{3}$$

In this equation, we call the variance term the *uncertainty* over the Q-values. As shown in the above formulation, this uncertainty depends on σ, σ_ϵ, α, batch size D and the frequency of the posterior updates. Specifically, higher values of σ and σ_ϵ and α lead to greater uncertainty, and higher values of D and more frequent posterior updates lead to lower uncertainty.

2.6 Hyperparameter Sensitivity

RL algorithms are known to be brittle and highly sensitive to their hyperparameters and the environments they run in. Adkins, Bowling, and White [1] suggest a framework for evaluating this hyperparameter sensitivity. The framework requires computing the expected performance of an algorithm $\omega \in \Omega$, which is defined as $\hat{p}(\omega, e, h) \doteq \frac{1}{|\mathcal{K}|}\sum_{\kappa \in \mathcal{K}} p(\omega, e, h, \kappa)$ where $\mathcal{K} \subset \mathbf{N}$ is the set seeds of random number generator, $e \in \mathcal{E}$ is an environment, and $h \in H^\omega$ is a hyperparameter setting.

After conducting a large number of runs across different algorithms, environments and hyperparameter settings, we find for each environment e, the 5th percentile $p_5(e)$ and the 95th percentile $p_{95}(e)$ of the distribution of observed performance for environment e. Then, the normalized environment score Γ is given by

$$\Gamma(\omega, e, h) \doteq \frac{\hat{p}(\omega, e, h) - p_5(e)}{p_{95}(e) - p_5(e)} \tag{4}$$

Finally, the hyperparameter sensitivity score of algorithm ω is defined as

$$\Phi(\omega) \doteq \frac{1}{|\mathcal{E}|}\sum_{e \in \mathcal{E}} \max_{h \in H^\omega} \Gamma(\omega, e, h) - \max_{h \in H^\omega} \frac{1}{|\mathcal{E}|}\sum_{e \in \mathcal{E}} \Gamma(\omega, e, h) \tag{5}$$

where the first term of the equation is called the per-environment tuned score and the second term is called the cross-environment tuned score.

In addition to the method by Adkins, Bowling, and White [1], several other papers provide useful perspectives on hyperparameter robustness. Obando-Ceron et al. [12] introduce a tuning hyperparameter consistency (THC) score to assess the transferability and consistency of hyperparameters across training regimes using a ranking algorithm, while Patterson et al. [15] use a four step procedure to choose a single best hyperparameter setting that is used for re-evaluation of the algorithm with a large number of environments. We decided to use Adkins, Bowling, and White [1] framework for its clear and simple scalar sensitivity measure.

2.7 Benchmark Environments

Multiple frameworks have been suggested to analyze the performance and efficiency of RL algorithms. Osband et al. [13] and Brockman et al. [5] provide a collection of scalable experiments that examine the capabilities of RL algorithms and highlight issues in their design – such as issues in generalization and exploration capabilities.

The environments provided include Cart Pole, a classic benchmark in RL that is based on the problem described in Barto, Sutton, and Anderson [3]. In this environment, the agent is tasked with stabilizing an upright pole mounted on a cart by applying discrete forces to shift the cart left or right. The environment is considered solved when the agent can keep the pole balanced for a predefined number of time steps without the pole falling or the cart moving out of bounds. Each time step in which the pole remains upright yields a reward that encourages the agent to maximize the duration of balance. The Cart Pole environment focuses on control and decision-making under constraints, making it an effective testbed for evaluating RL algorithms. As the agent must make only small adjustments to keep the pole upright, attempts at exploration often lead to failure, and are thus penalized.

In contrast to Cart Pole, Deep Sea [13] is a problem aimed at assessing the exploration capabilities of RL algorithms. In this problem, we consider an environment structured as a grid $N \times N$, where a block consistently starts in the upper left corner. At each time step, the agent may move the block either "left" or "right." Moving "right" incurs a small cost of $-0.01/N$, while moving "left" is cost-free. A significant reward of $+1$ is granted only if the agent selects "right" for exactly N consecutive steps, thereby reaching the bottom-right corner. This constitutes the sole rewarding trajectory; all other action sequences yield zero or negative returns. As such, the environment poses a challenging exploration problem due to the sparse and deceptive reward structure. The difficulty of this task becomes evident when considering that the probability of reaching the reward using uniformly random actions is 2^{-N}. For a grid size of 10×10, even naive random exploration is expected to quickly succeed. However, larger environments, such as 20×20 or more, require more efficient exploration strategies to find the reward.

Besides the classic RL environments, RL algorithms may also be assessed within the framework of a contextual bandits, an extension of the multi-armed bandit (MAB) problem. The MAB framework represents a scenario where an agent repeatedly selects an action from a set of available actions ("arms"), each yielding a reward from an unknown probability distribution. The main objective is to maximize cumulative rewards over time by strategically balancing exploration and exploitation. In the contextual bandit setting, the agent receives an additional context vector, which provides relevant information about the current situation. Using these contextual data alongside previous experience, the agent selects actions aimed at optimizing its overall reward. Contextual bandits provide a simple setting in which we can apply RL exploration strategies to better understand their behavior.

The MNIST environment [13, 9] can be framed as a contextual bandit problem, where each image from the MNIST dataset represents a context, and the agent must select an action corresponding to one of the ten-digit classes (0 through 9). Upon taking an action, the agent receives a binary reward: a reward of 1 if the chosen action matches the true label of the digit, and -1 otherwise. Unlike traditional RL tasks, there is no state transition or temporal component; each decision is made independently based on the observed context. This setup evaluates an agent's ability to learn from high-dimensional observations and map them effectively to optimal actions in a one-step decision-making framework.

3 Methodology

To investigate the performance, hyperparameter sensitivity, and transferability of BDQN, we designed a series of experiments using both contextual bandit environments and traditional RL benchmarks. This section details the implementation of BDQN, the environments used, the baselines for comparison, and the design of our sensitivity and transferability analyses.

We implemented the BDQN algorithm based on the specifications outlined by Azizzadenesheli and Anandkumar [2]. The implementation builds upon the CleanRL DQN implementation in JAX [8], with the adjustment of using DDQN and replacing the final linear layer of the Q-network with a BLR layer. Furthermore, the original implementation of the paper uses the notation of σ and σ_ϵ in the calculation of Cov_a rather than σ^2 and σ_ϵ^2, while multiplying Cov_a by σ. This appears to deviate from the expression given by Equation (2). In our implementation, we stick to the terms given by the equation.

To gain a thorough understanding of the performance of BDQN across different tasks, we use the environments outlined in Section 2.7, each designed to test a different aspect of the algorithm. The Deep Sea environment [13] is used at two difficulty levels, 10×10 and 20×20, to evaluate the algorithm in exploration-heavy settings. This environment is particularly well-suited for isolating the effect of exploration strategies, as it is possible to increase the complexity of exploration required without altering other environmental factors. Additionally, the Cart Pole [5] environment is used to evaluate BDQN's performance on tasks that do not typically require extensive exploration, helping to identify whether the algorithm is able to limit exploration when needed. Finally, MNIST [13] is framed as a contextual bandit problem to assess BDQN in a setting with no state transitions. In this setting, the reward is observed immediately, and prolonged exploration is effectively penalized, making it an environment well suited to examine how effectively BDQN exploits when sufficient experience is available.

We compare BDQN with a baseline DQN with ϵ-greedy exploration. To ensure a fair comparison, both models share the same architecture (excluding the BLR layer). Most hyperparameters (e.g., learning rate, replay buffer size, target update frequency) are kept constant across models, except for hyperparameters directly related to the BLR layer or exploration. The BLR hyperparameters are the forgetting factor (α), the batch size for the posterior update (*batch size*), the frequency of the posterior update (*update frequency*), the frequency of weight sampling (*weight sampling frequency*), the standard deviation of the prior (σ), the standard deviation of the noise (σ_ϵ) - six hyperparameters in total. For the exploration hyperparameters of ϵ-greedy, we consider the starting ϵ for exploration (*start* ϵ), the ending ϵ for exploration (*end* ϵ) and the fraction of total time steps it takes to get from start ϵ to end ϵ (*exploration fraction*) - three hyperparameters in total.

The primary evaluation metric that we use to compare different algorithms and hyperparameter settings is the average cumulative reward, which we call *score* in this paper. Additionally, we monitor the average Q-values of sampled experiences (from replay buffer) and the Q-values of the chosen actions. To better understand the behavior of BDQN beyond the raw episodic rewards, we also track the number of exploration steps taken, defined as the steps where the selected action is different from the action that would have been selected using the expected Q-values instead of the sampled Q-values. We monitor the uncertainty over Q-values as defined by the variance in Equation (3). To complement that, we compute the difference between the expected Q-values and the sampled Q-values, since uncertainty can be difficult to interpret when $\phi(x)$ is not normalized. We call this metric the Q-difference.

To measure the impact of BLR-specific hyperparameters on the performance and exploration of BDQN we perform hyperparameter sweeps, as described in Section 4. We also adopt the framework proposed by Adkins, Bowling, and White [1] to compare hyperparameter sensitivity of BDQN to the ϵ-greedy baseline.

All code, hyperparameter settings, and random seeds used and discussed in the following

Section 4 publicly available on github[1]. All experiments were executed in a WSL2 environment with Ubuntu 22.04.5 LTS on Windows 11 system, with AMD Ryzen 7 5800U CPU and 16GB RAM.

4 Experiments and Results

For the purpose of analyzing the exploration strategy and performance of BDQN we tested it on the three types of environments outlined in Section 2.7 and 3. For both algorithms, BDQN and ϵ-greedy, we started by running a hyperparameter sweep for all the exploration-relevant hyperparameters specified in Section 3, to identify sound hyperparameter values using Bayesian optimization with 20 iterations.

Then, we perform one-at-a-time hyperparameter tuning for BDQN's BLR hyperparameters to identify their impact on exploration. For each of the six hyperparameters, we used three different values, based on our findings in the initial sweep and the values reported by Azizzadenesheli and Anandkumar [2]. For each hyperparameter setting, five RNG seeds were used – all hyperparameter values are given in Appendix A. MNIST and Cart Pole were run for 500K steps, while Deep Sea environments were run for 1M steps. The results are given by environment.

4.1 MNIST

To ensure comparability between environments, we opted for a fully connected neural network rather than a convolutional neural network architecture, which is typically more suitable for MNIST [9]. In this setup, each input image of size 28×28 was flattened to a one-dimensional vector. This allowed us to match the architecture used in the other two environments.

When configured with a small forgetting factor ($\alpha = 0.01$), BDQN successfully "classified" 50.19% of the MNIST images in the last 10,000 steps, significantly exceeding the 10% success rate expected from random guessing and close to the 52.25% average achieved by the top five DQN agents identified by Bayesian optimization. However, despite running a considerably larger number of BDQN agents, the best agent found used ϵ-greedy DQN.

Importantly, only the smallest value of α led to a clear trend of decreasing uncertainty and Q-difference. In contrast, higher values of α (0.5 and 0.9) resulted in higher uncertainty, more exploration, and poorer performance. In particular, the number of exploration steps for the higher α values was on average 50.89% of the total number of steps.

Additionally, we observed a consistent trend when varying the batch size. Larger batch sizes generally produced lower uncertainty, fewer exploration steps and higher scores. The largest batch size (5000) yielded the highest average score and the fewest exploration steps (7.69% of the total number of steps). To a lesser extent, we observed that, similarly to batch size, more frequent updates to the posterior (500 steps per update) also reduced uncertainty, curbed exploration, and achieved higher rewards. At the same time, no consistent trend was observed by tuning the frequencies of weight sampling.

Adjusting the prior standard deviation σ, we found that a lower value ($\sigma = 0.1$) reduced uncertainty and exploration – an effect usually linked to improved performance in this environment – yet, in this instance, it resulted in lower scores. Regarding the noise parameter

[1] https://github.com/sagisch/RP-BDQN

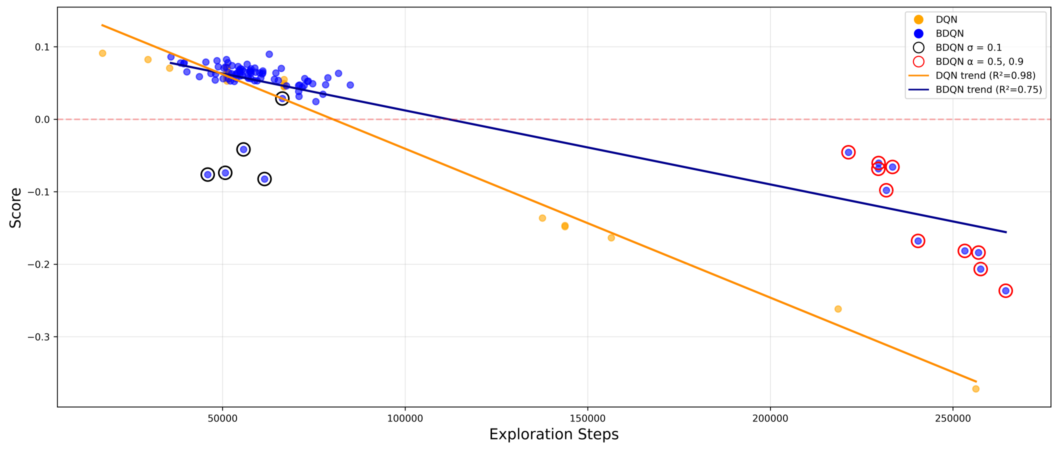


Figure 1: Scores of all BDQN and DQN runs as a function of exploration steps in MNIST. High values of α and the lowest values of σ are result in outliers which are highlighted in the plot.

σ_ϵ, lower values consistently resulted in less uncertainty, reduced exploration, and improved overall scores.

Figure 1 compares the scores of both BDQN and DQN as a function of exploration steps. As shown in the graph, both BDQN and DQN achieved higher scores when less exploration steps were taken. The fitted lines highlight that, in our experiments, the decrease in score with increasing exploration steps follows a linear trend. Additionally, the graph shows that higher values of α in BDQN lead to more exploration. As discussed, although σ does not appear to affect exploration, it has a negative impact on performance. All BDQN data points that are not highlighted with a circle have the default $\alpha = 0.01$ and $\sigma = 0.5, 1$. The fact that these instances achieved the best scores supports the observation that lower α and higher σ were preferred in this environment.

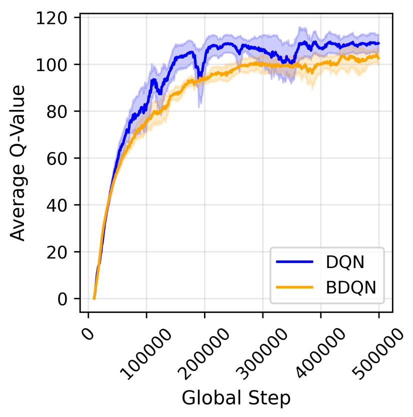


Figure 2: Average learned Q-values of sampled experiences from replay buffer of BDQN and DQN in Cart Pole with 95% CI.

4.2 Cart Pole

Overall, trends similar to those observed in MNIST were also identified in the Cart Pole environment. In particular, Cart Pole benefits from minimal exploration. The best run of BDQN outperformed the best run of ϵ-greedy DQN, however, given that most of our hyperparameter tuning effort was focused on BDQN, this result is not surprising. Figure 2 shows that despite the best run of BDQN outperforming the best run in DQN, the Q-values learned by DQN were generally slightly higher, which may suggest that the ϵ-greedy strategy may still be advantageous in the Cart Pole setting. Nevertheless, we did not find a significant difference in the average performance between the two algorithms (similar plots of the learned Q-values in other environments can be found in Appendix C).

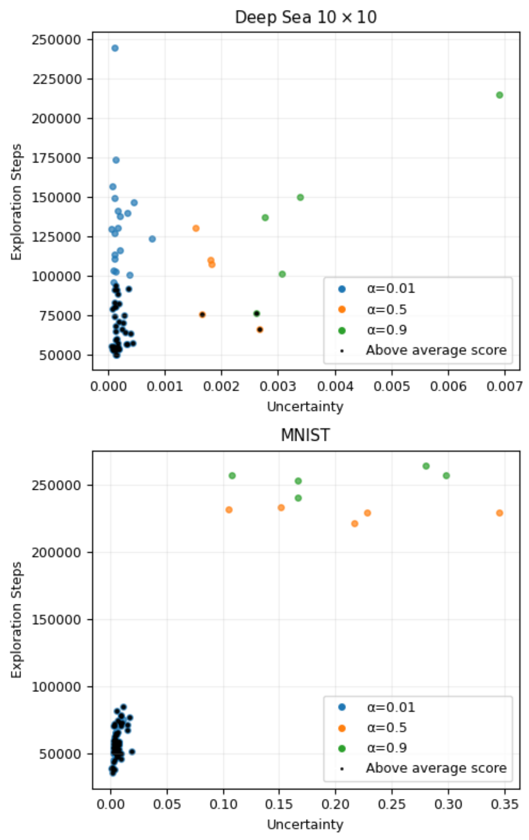


Figure 3: Exploration as a function of uncertainty of all runs in tested environment. Different values of α are highlighted, along with runs with an above average score which are filled with a dark dot.

During the hyperparameter tuning of BDQN, we observed that lower values of α led to reduced uncertainty, decreased exploration, and improved scores. While the influence of α on uncertainty and exploration was less pronounced than in the MNIST environment, its impact on performance was still the most significant among all tested hyperparameters. Figure 3 shows that although higher values of α reduced uncertainty and encouraged exploration in Cart Pole, the change was not as dramatic as in MNIST. Similarly to α, increasing the batch size also reduced uncertainty and exploration, resulting in better scores. The posterior update frequency had a comparable effect, though to a lesser extent. In contrast, weight sampling frequency did not show a clear impact on uncertainty, exploration, or performance – although the most frequent sampling setting did yield a slightly higher score.

The effect of σ on uncertainty, exploration, and reward was not consistent, but the highest value of σ did achieve a slightly better score. On the other hand, σ_ϵ had a clearer influence: larger values increased uncertainty and exploration. Interestingly, it was the intermediate value of σ_ϵ that resulted in the highest score.

4.3 Deep Sea

Expectedly, the performance of the algorithms varied substantially between Deep Sea grid size 10×10 and 20×20.

Deep Sea 10×10 was solved successfully by both BDQN and ϵ-greedy DQN. However, under the hyperparameter settings used, BDQN appeared to have an advantage over DQN. The best runs of BDQN achieved over a 10% improvement in score compared to DQN.

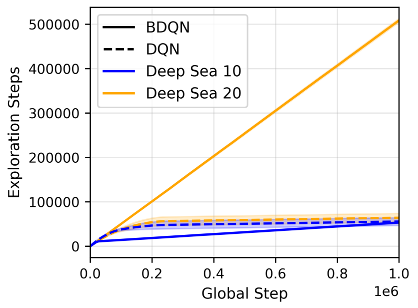


Figure 4: Comparison of BDQN and DQN exploration in Deep Sea environments with 95% CI.

Similarly to the other environments we covered, in Deep Sea 10×10 environment, a lower value of α led to reduced uncertainty, less exploration, and higher scores as well. Figure 3 shows that Deep Sea 10×10 does in fact benefit from less exploration, having all of its runs that achieve an above average score concentrated at the bottom of the plot, with a lower number of exploration steps. Additional plots demonstrating this trend can be found in Appendix B. Besides α, larger batch sizes also reduced uncertainty; however, their relationship with exploration and rewards was not monotonic. Interestingly, an intermediate batch size resulted in the least exploration and the highest score. The frequency of posterior updates and weight sampling did not show a consistent trend with respect to uncertainty, exploration, or rewards. Again, the intermediate setting yielded the best performance, with the lowest level of exploration and the highest score.

Contrary to what the uncertainty formula might suggest, we did not observe a clear relationship between σ and uncertainty (the intermediate value of σ once more resulted in the least exploration and highest score). In contrast, σ_ϵ behaved more predictably – lower values led to reduced uncertainty and smaller Q-value differences. The lowest σ_ϵ also resulted in the least exploration steps and the highest score.

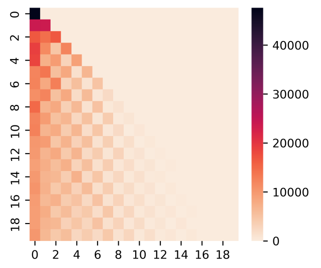


Figure 5: BDQN state coverage in Deep Sea of size 20×20.

In Deep Sea 20×20 the results were considerably different. In the larger Deep Sea environment, ϵ-greedy DQN fails completely. In fact, it did not reach the reward at position (19, 19) even once, while BDQN with the best hyperparameter setting found reached it 16.4 times on average (over all seeds). Figure 4 shows that the number of exploration steps taken by BDQN dramatically exceeds the one taken by DQN, but at the same time it explores even less than DQN in a simpler environment as Deep Sea 10×10. Additionally, Figures 5 and 6 show heatmaps representing the state coverage of both algorithms, which clearly demonstrate how ϵ-greedy DQN under-explores the state space.

Although a small α (i.e., 0.01) resulted in significantly lower uncertainty and smaller Q-differences, as in the other environments, it came at the cost of exploration – crucial for the larger environment. In this environment, the highest value of α (0.9) produced the greatest uncertainty, led to the most exploration, and resulted in rewards being reached significantly more often than with lower α values. We observed a monotonic relationship: as α increased, so did uncertainty, exploration steps and scores. Figure 3 shows that Deep Sea 20×20 does benefit from more exploration steps, as most of its best performing runs are concentrated in the upper-right corner of the plot (runs which performed the most exploration).

While larger batch sizes also reduced uncertainty, the use of a low default α likely limited the agents' ability to discover rewards, potentially masking their full effect. Variations in posterior update frequency and weight sampling frequency did not show any noticeable impact on uncertainty, exploration, or reward. However, this lack of effect may be due to the low default α used in these experiments.

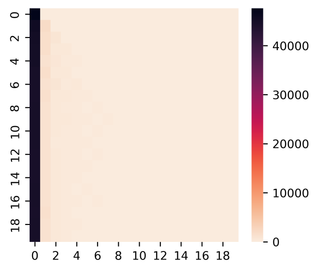


Figure 6: DQN state coverage in Deep Sea of size 20×20.

Additionally, although increasing σ did not substantially raise uncertainty over time as would be expected, it did lead to a higher average number of rewards reached. Finally, higher values of σ_ϵ increased uncertainty and resulted in a slight improvement in reward, with the highest value ($\sigma_\epsilon = 1.2$) leading to an average of one reward reached.

Another observation that was made in BDQN across *all* environments is that the function of exploration steps against the total number of steps in the environment tends to be linear, or piecewise linear, as demonstrated in Figure 4. This means that BDQN has an approximately constant probability of exploring in these regions. In the most challenging environment we tried (Deep Sea 20×20), that probability was around 1/2, meaning that around half of the steps were exploration steps. Again, we refer to Figure 3 which shows the number of exploration steps of BDQN across all environments, as a function of uncertainty. As shown, in all environments, exploration appear to be capped at approximately 50% of the total steps (Cart Pole is the only environment which does not reach that). Furthermore, the figure makes it clear that most environments, except for Deep Sea 20×20, benefit from less exploration, which is achieved by smaller values of α. To clearify, the dark dots inside the data points represent runs with an above average score. As shown, they are typically concentrated in the bottom-left corner where uncertainty is low and exploration is minimal – a trend that is reversed in Deep Sea 20×20.

Furthermore, we computed hyperparameter sensitivity scores for BDQN and ϵ-greedy DQN. BDQN achieved a sensitivity score of 0.2810 (with a per-environment tuned score of 1.14), while DQN scored 0. This result is not unexpected. Because we used Bayesian optimization to tune DQN, each environment ended up being tested with a different set of hyperparameters. Out of the $3^3 = 27$ possible hyperparameter settings of DQN, only a two were shared across all environments. Although the lack of common hyperparameter configurations across environments is itself an indicator of high sensitivity, it also means we were unable to compute a reliable sensitivity score for DQN.

Lastly, we include the optimal hyperparameter settings we found in all environments in Table 1. This table consists of the preferable hyperparameter values, when all other hyperparameters are kept constant in their default values. This should not be interpreted as the best hyperparameter combinations.

Environment	α	Posterior Updates	Batch Size	Weight Sampling	σ	σ_ϵ
Deep Sea 20×20	0.9	2000	2000	100	1	1.2
Deep Sea 10×10	0.01	1000	2000	100	0.5	0.8
Cart Pole	0.01	500	5000	50	1	1
MNIST	0.01	500	5000	500	1	0.8

Table 1: Best BDQN hyperparameter values in different environments.

5 Discussion

Our experiments demonstrate that BDQN's effectiveness is environment-dependent. It provides substantial improvements over ϵ-greedy DQN in exploration-heavy tasks like Deep Sea at larger grid sizes, but offers only marginal or no improvements in simpler environments like MNIST and Cart Pole. This suggests that BDQN is particularly well-suited for environments with sparse or deceptive reward structures. It is also important to note that although BDQN achieved better performance than ϵ-greedy DQN in Deep Sea 20×20, BDQN success was still very limited to reaching the final reward only a relatively few times with one million steps and the learned Q-values did not converge to the appropriate values (shown in Appedix C).

Among the BLR hyperparameters, the forgetting factor α emerged as the most influential in all environments. A lower value of α (i.e., 0.01) typically suppressed exploration and improved performance in simpler tasks. In contrast, higher α values (i.e., 0.5 and 0.9) increased exploration and in Deep Sea 20×20, where more exploration was necessary, it resulted in better performance. Other hyperparameters, such as batch size and σ_ϵ, generally behaved as expected: larger batches and lower noise variances reduced uncertainty and exploration, often leading to improved performance in simpler environments. However, some effects were too weak or inconsistent across environments to support definitive conclusions. This aligns with the optimal hyperparameter values summarized in Table 1, which show that optimal α values correspond to the complexity of the environment, and that larger batch sizes generally outperform smaller ones. In contrast, the effects of other hyperparameters appear less predictable.

Another notable observation is that BDQN's exploration tends to persist throughout training, with a relatively constant probability of exploration over exploitation, even as uncertainty decreases. In none of our experiments did exploration cease entirely. This behavior may be advantageous in environments where rewards are sparse and continued exploration is essential throughout training. However, it may be less beneficial in tasks where the optimal policy is discovered early, as BDQN continues to explore suboptimal transitions that hurt overall performance.

6 Limitations and Future Work

This study presents several limitations that should be acknowledged. First, the experimental evaluation was restricted to three environments – MNIST, Cart Pole, and Deep Sea – which, while covering a range of complexity and exploration demands, are not sufficient to generalize the findings across the broader landscape of RL problems. A more comprehensive evaluation across diverse environments would be necessary to confirm the robustness and generalizability of the results. Second, hyperparameter tuning efforts were primarily focused on BDQN, whereas the ϵ-greedy DQN baseline received comparatively less attention. As a result, the performance gap observed between the two methods may partially reflect differences in tuning rather than intrinsic algorithmic benefits. Lastly, the hyperparameter sensitivity analysis employed a relatively new framework proposed by Adkins, Bowling, and White [1] which has not yet seen widespread use in practice. Consequently, the sensitivity scores reported in this work should be interpreted with caution. More importantly, we believe that the hyperparameter tuning in our experiments was not sufficiently extensive to yield reliable sensitivity scores. In contrast, the original paper conducted 200 runs for each of five environments across 625 hyperparameter configurations, far exceeding the relatively modest number of runs we used in our study (a figure comparing BDQN's computed hyperparameter sensitivity score to the algorithms tested by Adkins, Bowling, and White [1] can be found in Appendix D).

Future research should aim to evaluate BDQN across a broader range of environments and real-world contextual bandit applications. This would help assess the generalizability of the findings beyond the three environments explored in this work. Additionally, a more extensive hyperparameter sensitivity analysis involving a larger number of hyperparameter settings and runs could yield deeper insights into the algorithm's behavior and the effect of its hyperparameters, particularly for hyperparameters that did not exhibit consistent patterns in our results. Finally, it may be valuable to explore hybrid approaches that combine BDQN with adaptive ϵ-scheduling or policy-switching mechanisms to reduce persistant exploration in simpler environments while retaining BDQN's strengths in complex, sparse-reward settings.

7 Conclusions

This work investigated the effectiveness of Bayesian Deep Q-Networks (BDQN) in environments with varying exploration complexity and addressed three core research questions: performance, sensitivity, and transferability. First, we systematically compared BDQN to ϵ-greedy DQN across both sparse-reward and dense-reward tasks, finding that BDQN outperforms standard DQN in exploration-heavy settings such as Deep Sea 20×20 while having no consistent advantage over DQN in simpler environments, thus answering the performance question. Second, we identified the forgetting factor (α) as the most influential exploration-related hyperparameter, revealing that BDQN is highly sensitive to α, while other BLR hyperparameters had more moderate effects. We also detailed the effects of each hyperparameter, providing insight into how they shape BDQN's exploratory behavior – thereby addressing the sensitivity question. Third, we showed that BDQN performance is highly task-dependent and requires careful hyperparameter tuning. Using BDQN across different types of tasks may not be trivial, as the optimal hyperparameter values can vary significantly, especially between exploration-heavy and simpler tasks, answering the transferability question. These findings deepen our understanding of BDQN's exploration strategy and offer practical guidance for its use in reinforcement learning.

8 Responsible Research

This work adheres to the principles of the Netherlands Code of Conduct for Research Integrity: honesty, scrupulousness, transparency, independence, and responsibility.

Honesty has been upheld throughout this research by reporting experimental procedures and results accurately, avoiding overstated claims, and presenting both the strengths and limitations of the algorithms studied. We have not attempted to exaggerate the findings or selectively report favorable results.

Scrupulousness was ensured by designing and executing our experiments with methodological rigor. All algorithms were evaluated using consistent experiments, including multiple random seeds to account for randomness and measure the variability in the results. Performance metrics were selected to reliably capture algorithm behavior, and confidence intervals are reported where appropriate.

Transparency is supported by the full disclosure of our experimental setup. Every step of the research process is documented in detail to enable replication. This includes hyperparameter values, random seeds and type of machine used to run the code. The source code and data used to generate all the figures and results are publicly available in a dedicated GitHub repository, enabling full verification and reuse by other researchers.

Independence was maintained by focusing purely on the properties and empirical behavior of the reinforcement learning exploration strategies studied, without influence from commercial or political stakeholders.

Responsibility has been considered by ensuring that the research remains relevant to the scientific community. Furthermore, we recognize that reinforcement learning can have broader societal implications. In this research, we focus solely on algorithmic evaluation, rather than application domains, to develop general scientific understanding in a responsible and neutral manner.

Together, these measures address reproducibility and integrity challenges in a responsible, rigorous and ethical way.

References

[1] Jacob Adkins, Michael Bowling, and Adam White. *A Method for Evaluating Hyperparameter Sensitivity in Reinforcement Learning.* arXiv:2412.07165 [cs]. Feb. 2025. DOI: 10.48550/arXiv.2412.07165. URL: http://arxiv.org/abs/2412.07165 (visited on 05/02/2025).

[2] Kamyar Azizzadenesheli and Animashree Anandkumar. *Efficient Exploration through Bayesian Deep Q-Networks.* arXiv:1802.04412 [cs]. Sept. 2019. DOI: 10.48550/arXiv.1802.04412. URL: http://arxiv.org/abs/1802.04412 (visited on 04/21/2025).

[3] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846. DOI: 10.1109/TSMC.1983.6313077.

[4] Djallel Bouneffouf and Irina Rish. *A Survey on Practical Applications of Multi-Armed and Contextual Bandits.* arXiv:1904.10040 [cs]. Apr. 2019. DOI: 10.48550/arXiv.1904.10040. URL: http://arxiv.org/abs/1904.10040 (visited on 05/02/2025).

[5] Greg Brockman et al. *OpenAI Gym.* arXiv:1606.01540 [cs]. June 2016. DOI: 10.48550/arXiv.1606.01540. URL: http://arxiv.org/abs/1606.01540 (visited on 06/02/2025).

[6] Meire Fortunato et al. *Noisy Networks for Exploration.* arXiv:1706.10295 [cs]. July 2019. DOI: 10.48550/arXiv.1706.10295. URL: http://arxiv.org/abs/1706.10295 (visited on 05/02/2025).

[7] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning.* arXiv:1509.06461 [cs]. Dec. 2015. DOI: 10.48550/arXiv.1509.06461. URL: http://arxiv.org/abs/1509.06461 (visited on 04/29/2025).

[8] Shengyi Huang et al. *CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms.* arXiv:2111.08819 [cs]. Nov. 2021. DOI: 10.48550/arXiv.2111.08819. URL: http://arxiv.org/abs/2111.08819 (visited on 07/06/2025).

[9] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 00189219. DOI: 10.1109/5.726791. URL: http://ieeexplore.ieee.org/document/726791/ (visited on 06/02/2025).

[10] Long-Ji Lin. "Reinforcement learning for robots using neural networks". UMI Order No. GAX93-22750. PhD thesis. USA: Carnegie Mellon University, 1992.

[11] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning.* arXiv:1312.5602 [cs]. Dec. 2013. DOI: 10.48550/arXiv.1312.5602. URL: http://arxiv.org/abs/1312.5602 (visited on 04/27/2025).

[12] Johan Obando-Ceron et al. *On the consistency of hyper-parameter selection in value-based deep reinforcement learning.* arXiv:2406.17523 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2406.17523. URL: http://arxiv.org/abs/2406.17523 (visited on 07/06/2025).

[13] Ian Osband et al. *Behaviour Suite for Reinforcement Learning.* arXiv:1908.03568 [cs]. Feb. 2020. DOI: 10.48550/arXiv.1908.03568. URL: http://arxiv.org/abs/1908.03568 (visited on 06/02/2025).

[14] Ian Osband et al. *Deep Exploration via Bootstrapped DQN.* arXiv:1602.04621 [cs]. July 2016. DOI: 10.48550/arXiv.1602.04621. URL: http://arxiv.org/abs/1602.04621 (visited on 04/27/2025).

[15] Andrew Patterson et al. *The Cross-environment Hyperparameter Setting Benchmark for Reinforcement Learning.* arXiv:2407.18840 [cs]. July 2024. DOI: 10.48550/arXiv.2407.18840. URL: http://arxiv.org/abs/2407.18840 (visited on 07/06/2025).

[16] Richard S. Sutton and Andrew Barto. *Reinforcement learning: an introduction.* eng. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press, 2020. ISBN: 978-0-262-03924-6.

[17] William R. Thompson. "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: *Biometrika* 25.3/4 (Dec. 1933), p. 285. ISSN: 00063444. DOI: 10.2307/2332286. URL: https://www.jstor.org/stable/2332286?origin=crossref (visited on 06/09/2025).

[18] William R. Thompson. "On the Theory of Apportionment". In: *American Journal of Mathematics* 57.2 (Apr. 1935), p. 450. ISSN: 00029327. DOI: 10.2307/2371219. URL: https://www.jstor.org/stable/2371219?origin=crossref (visited on 06/09/2025).

[19] Christopher Watkins. "Learning From Delayed Rewards". PhD thesis. King's College, 1989.

A Hyperparameter Tuning

The tables below summarize the hyperparameter values used for hyperparameter tuning. For BDQN, each of the hyperparameters was changed at a time, while keeping the others fixed at a default value. The values are given in Table 2. Each setting was run with five different random seeds. For DQN, Bayesian optimization was used to identify the best set of hyperparameters from the values given in Table 3.

Hyperparameter	**Default Value**	**Search Space**
α (forgetting factor)	0.01	0.01, 0.5, 1.0
Posterior Update	1000	500, 1000, 2000
Batch Size	2000	1000, 2000, 5000
Weight Sampling	100	50, 100, 500
σ (prior std)	0.5	0.1, 0.5, 1.0
σ_ϵ (noise std)	0.8	0.8, 1.0, 1.2

Table 2: BDQN hyperparameter values used in tuning.

Hyperparameter	**Search Space**
Start ϵ	0.5, 0.75, 1.0
End ϵ	0.01, 0.25, 0.5
Exploration Fraction	0.1, 0.25, 0.5

Table 3: DQN hyperparameter values used in Bayesian optimization.

B Exploration and Rewards in Deep Sea

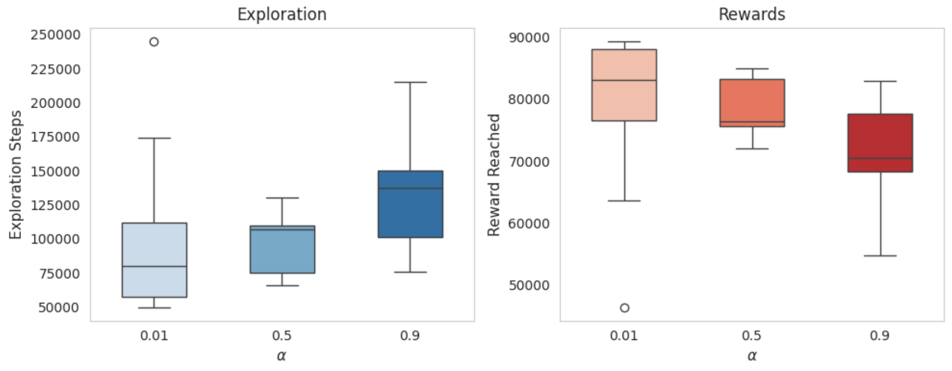


Figure 7: Exploration and Rewards of BDQN in Deep Sea 10×10, showing the environment benefits from little exploration.

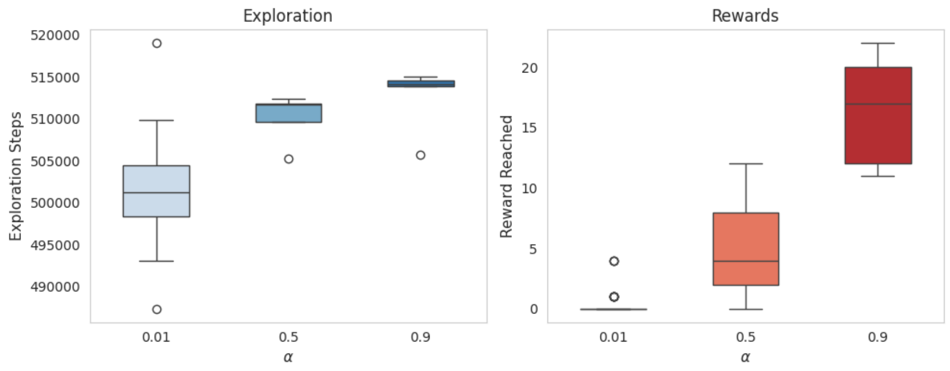
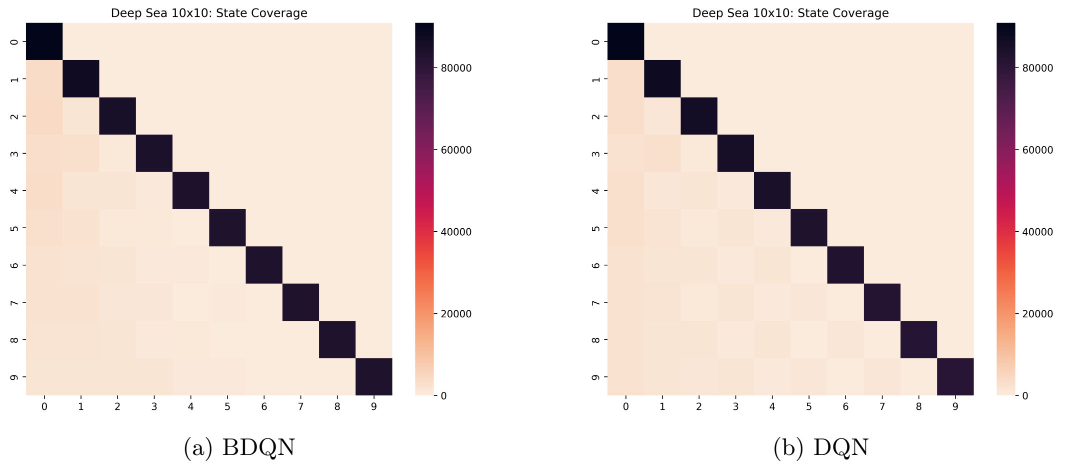


Figure 8: Exploration and Rewards of BDQN in Deep Sea 20×20, showing the environment benefits from additional exploration.



(a) BDQN

(b) DQN

Figure 9: Comparison of BDQN and DQN state coverage in Deep Sea of size 10×10

C Q-Values Learning in BDQN and DQN

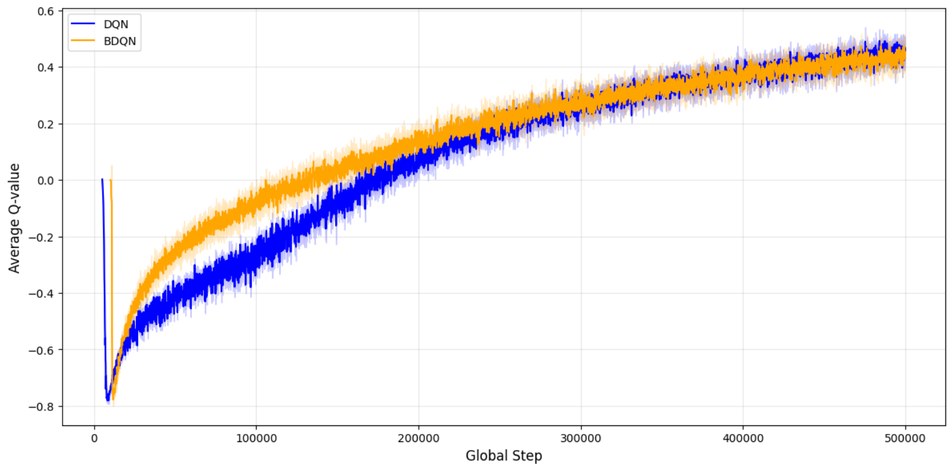


Figure 10: Average learned Q-values of sampled experiences from replay buffer of BDQN and DQN in MNIST with 95% CI.

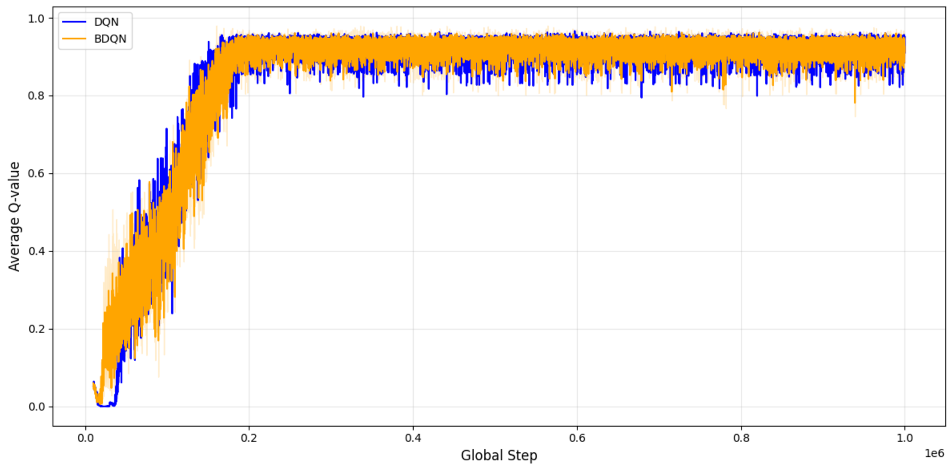


Figure 11: Average learned Q-values of sampled experiences from replay buffer of BDQN and DQN in Deep Sea 10×10 with 95% CI.

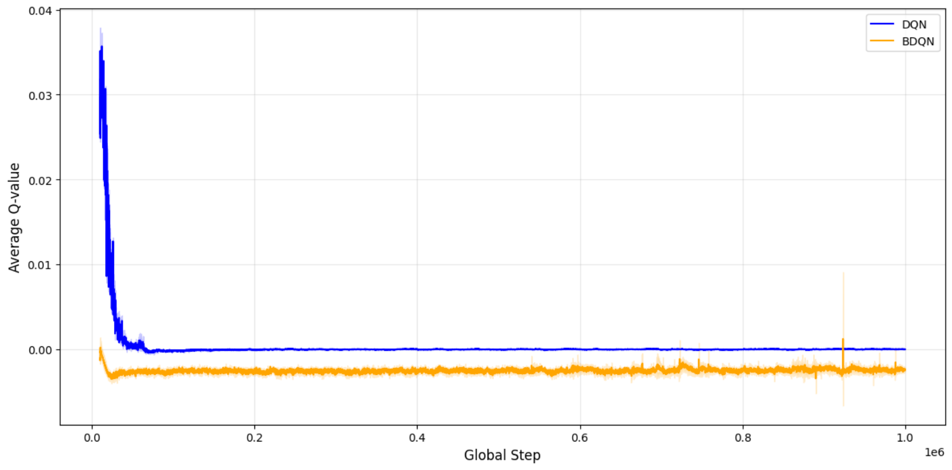


Figure 12: Average learned Q-values of sampled experiences from replay buffer of BDQN and DQN in Deep Sea 10×10 with 95% CI. This plot shows that neither of the algorithms converge to the appropriate Q-values, however BDQN manages to reach the reward a few times due to vast exploration.

D Hyperparameter Sensitivity

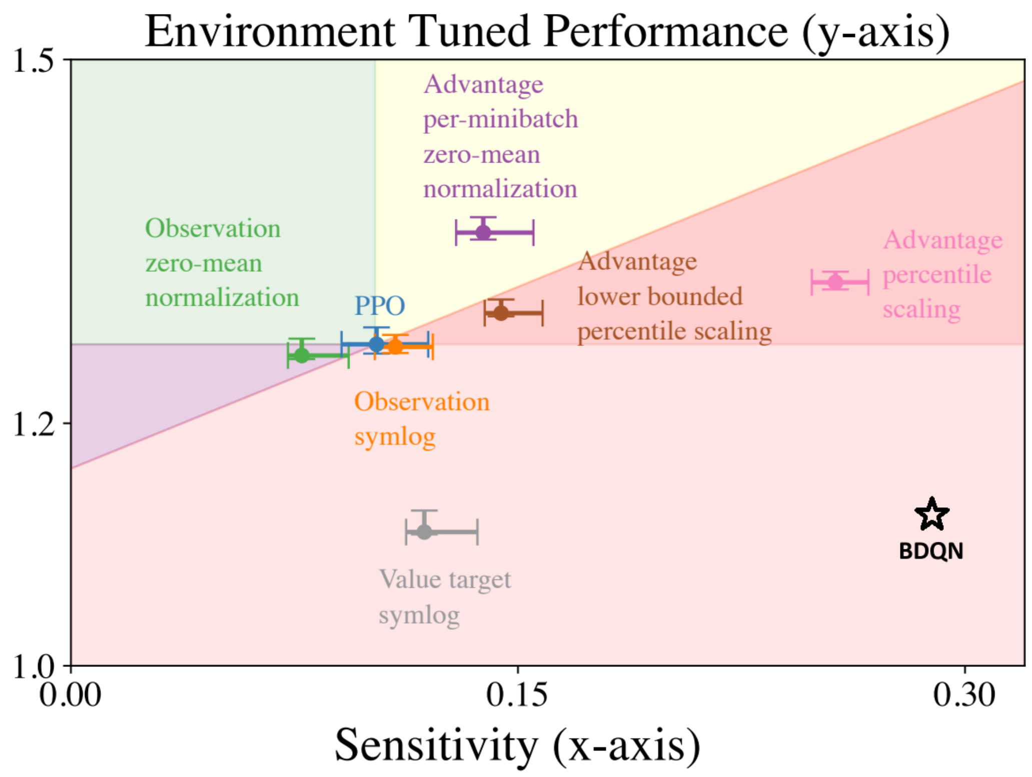


Figure 13: BDQN hyperparameter sensitivity embedded in the original figure published in Adkins, Bowling, and White [1].