Revisiting Langevin Monte Carlo Applied to Deep Q-Learning: An Empirical Study of Robustness and Sensitivity

Pablo Hendriks Bardaji[1]
Supervisor(s): 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
June 22, 2025

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

Abstract

Deep Reinforcement Learning has achieved superhuman performance in many tasks, such as robotic control or autonomous driving. Algorithms in Deep Reinforcement Learning still suffer from a sample efficiency problem, where, in many cases, millions of samples are needed to achieve good performance. Recently, Bayesian uncertainty-based algorithms have gained traction. This work focuses on providing a better understanding of the behaviour of Langevin Monte Carlo algorithms for Bayesian posterior approximation applied on top of Q-learning. This research builds on top of already existing algorithms, aiming to provide a better understanding of the underlying mechanics that drive them. We provide empirical experimentation with different hyperparameters in three different environments. Our results suggest that hyperparameters that were previously thought not to have a big impact on the algorithms are crucial for deep exploration.

1 Introduction

Reinforcement Learning (RL) is a subfield of machine learning focused on how an agent learns to make decisions by interacting with an environment to maximise a reward signal. At the intersection between RL and Deep Learning lies Deep Reinforcement Learning (DRL). By combining reinforcement learning with deep neural networks, DRL has achieved super-human performance in tasks ranging from Atari video games (Bellemare et al., 2013), to complex robotic control. Yet, sometimes millions of steps are required to achieve performance at the human level, highlighting significant challenges in sample efficiency.

One of the central problems of reinforcement learning is the dilemma between taking exploration or exploitation actions (Sutton and Barto, 2020). One of the more naïve approaches to this problem is ε-greedy, an exploration strategy that takes random actions with a probability ε. An alternative solution to this problem could be to design clever exploration methods in which uncertain and under-explored promising areas of the environment are actively visited to learn faster, instead of just randomly choosing an action. However, quantifying and using this uncertainty is not a trivial task.

To attempt to solve this, recent algorithms include: random network distillation (Burda et al., 2019), randomised prior functions (Osband et al., 2018), count-based exploration with hashing (Tang et al., 2017), noisy networks (Fortunato et al., 2018), and Monte-Carlo dropout (Gal and Ghahramani, 2016) - return intrinsic rewards or perturbed value estimates that favour novel states. These methods exhibit remarkable performance in hard-exploration benchmarks, though they tend to have a lot of different hyperparameters, and their sensitivity as well as performance in classical tasks, as bandits, remains less clear.

An interesting line of research proposes approximating the Bayesian posterior using Markov Chain Monte Carlo. The posterior offers inherent uncertainty quantification, and while in most cases it is not feasible to infer, algorithms can rely on approximations of it. The present study focuses on exploration via Langevin Monte Carlo (LMC) (Ishfaq et al., 2024). Concretely, the algorithms we focus on are the Langevin Monte Carlo Least-Squares Value Iteration (LMC-LSVI) algorithm, which performs noisy gradient descent updates, and the Adam Langevin Monte Carlo Deep Q-Network (Adam LMCDQN) algorithm, a variant of LMC-LSVI based on the Adam SGLD optimiser (Kingma and Ba, 2017; Ishfaq et al., 2024; Kim et al., 2020). We analyse the sensitivity to hyperparameters of these algorithms and compare them with ε-greedy.

As part of the research, the study will also provide insights into the following questions:

1. Motivated by the proven theoretical guarantees of LMC-LSVI, we will research how well Adam LMCDQN generalises to simpler contextual bandit settings.
2. Motivated by the design of LMC-LSVI and Adam LMCDQN, we will analyse the effect of changing the number of updates per batch of data.
3. Motivated by the theory behind LMC, the effects of different hyperparameter choices for the scale of the noise, the weight given to Adam in Adam LMCDQN and the learning rate will be analysed in both exploration-heavy environments and exploitation-heavy environments.

4. Motivated by the brittleness of DRL algorithms, we will analyse what is the set of hyperparameters that generalises best across environments.

The rest of the paper is structured as follows: First, in Section 2, the related work to our contribution is explained, followed by Section 3 where a comprehensible explanation of the topics necessary to understand our contribution is explained. We then explain the algorithms analysed in this paper in Section 4. After that, we delve into the experimental setup of our research in Section 5 to then show the results of the conducted experiments in Section 6. Then we discuss the responsibility of our research in Section 7. Finally, we conclude the paper in Section 8.

2 Related Work

Several epistemic-uncertainty exploration methods have been proposed. For example, there is a line of research that proposes rewarding states that have not been visited often. Tang et al. (2017) proposed Count-Based Exploration With Hashing and Burda et al. (2019) proposed Random Network Distillation; both of these methods give an intrinsic reward proportional to the novelty of a state. While methods in this line of research have shown remarkable performance in some tasks, in this research, we focus on the line of research that uses Bayesian posterior approximation.

Posterior Approximation is a popular approach when trying to use the epistemic uncertainty, Osband et al. (2013) pioneered using the posterior in Reinforcement Learning with Posterior Sampling for Reinforcement Learning. One of the traditional approaches to using uncertainty is to use Bayesian Neural Networks. Azizzadenesheli and Anandkumar (2019) proposed BDQN, where the last layer of the neural network is replaced with a Bayesian probabilistic regression; or Benatan and Pyzer-Knapp (2019) proposed a fully Bayesian Recurrent Neural Network for uncertainty quantification. Another of the first approaches was Randomised Least Squares Value Iteration (Russo, 2019; Osband et al., 2016b). This approach is provably efficient under the tabular MDP setting but does not generalise well to larger problems. This approach was extended to the DRL setting with Bootstraped DQN (Osband et al., 2016a), this approach estimates the posterior by training several independent networks. From this idea of bootstrapping different networks, several algorithms have appeared (Osband et al., 2018; Van Der Vaart et al., 2024). Another approach to posterior approximation is (Gal and Ghahramani, 2016), which proposed Monte Carlo Dropout to estimate the posterior.

Closer to our work is (Dwaracherla and Roy, 2021), who proposed the Langevin DQN algorithm, which performs noisy updates to parameters. With empirical studies, they demonstrated that the algorithm achieved deep exploration, though there was no experimentation in pixel-based environments.

Ishfaq et al. (2024) proposed the **LMC-LSVI** and **Adam LMCDQN** algorithms we analyse in this paper. They provided a theoretical analysis of the regret bound of LMC-LSVI under certain conditions, and motivated by those theoretical properties and ideas from AdamSGLD (Kim et al., 2020), they developed the Adam LMCDQN algorithm. Apart from the theoretical analysis, they provided extensive experimentation in pixel-based environments, namely, the Atari 2600 games. Given the similarities with LangevinDQN, they also provided an empirical comparison of AdamLMCDQN with it. They also showed some experimentation with different parameters in the Q*bert environment. We revisit and extend that experimentation with different environments and parameters.

3 Background

In this section, we will introduce the necessary background for our contribution. First, we explain Markov Decision Processes in 3.1, the foundation for Reinforcement Learning. Then, in 3.2 we introduce the Deep Q Network algorithm. After, we continue with an explanation of the idea behind the analysed algorithms in 3.3. Then we give an introduction to the Langevin Monte Carlo method in 3.4.

3.1 Markov Decission Process

A Markov Decision Process (MDP) is a mathematical framework used to model sequential decision problems (Bellman, 1957). Formally, it is defined by the tuple (S, A, T, R, γ), where S is the set of all possible states of the environment, A is the set of all possible actions, $T : S \times A \to S$ is the

transition function, R is the reward function $S \times A \to \mathbb{R}$, and $\gamma \in [0, 1]$ represents the discount factor. The MDP has the Markov property, which allows the model to focus only on the current state for decision making, simplifying analysis and computation.

In a standard RL setting, an agent observes a state s_t each time step t, and based on some policy $\pi : S \to A$, takes an action a_t and receives a reward $r_t = R(s_t, a_t)$. The goal is to find a policy π to maximise the expected cumulative discounted reward.

A key concept is the Q-function (or action value function), which quantifies the expected return when starting from state x at time step t, taking action a, and then following policy π until some time horizon T. Concretely, it is defined by

$$Q_t^\pi(x, a) = \mathbb{E}_\pi \left[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}(x_{t'}, a_{t'}) | \, x_t = x, \, a_t = a \right] \tag{1}$$

Based on this function, the Q-learning algorithm was proposed (Watkins, 1989). The goal of this algorithm is to learn the optimal action selection policy by repeatedly interacting with the environment without the need of having a model of it. The core Q-learning algorithm maintains a Q-table, consisting of Q-values for every pair of states and actions, and it updates them based on the Q-function. Essential to this update process is the Temporal Difference Loss, which captures the difference between the current Q-value estimate and the one-step Bellman target. Formally, it is defined by:

$$\text{TD Loss}(\theta) = E[(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) - Q(s_t, a_t; \theta))^2] \tag{2}$$

3.2 Deep Q Network

Based on the Q-function and following the Q-learning approach, the Deep Q Network (DQN) algorithm replaces the Q-table with Neural Networks (NNs). This approach allows handling of much higher-dimensional inputs such as images. Mnih et al. (2013) first proposed the idea of using NNs to estimate the Q-function, achieving better performance than a human expert in three Atari games. Mnih et al. (2015) refined the idea and achieved human-level comparable performance in 49 Atari games without fine-tuning the algorithm or its parameters. It also introduced two key ideas that are central for the empirical performance of DQN and the algorithms based on it:

- The experience buffer, which stores past interactions in a replay memory. This enables agents to learn from a diverse, decorrelated set of experiences rather than just the most recent trajectory. This mechanism stabilises learning, improves policy convergence, and mitigates forgetting past interactions (Liu and Zou, 2017).
- The target network, which maintains a delayed set of weights used to calculate targets for the Temporal Difference error. This technique decouples the changing Q estimates generated by the network, stabilising learning. By having a separate set of weights fixed for computing target values, it prevents feedback loops that can lead to the network chasing its estimates (Fan et al., 2020).

3.3 Posterior Sampling In Reinforcement Learning

Bayesian statistics is a statistical framework in which some initial beliefs are systematically updated with the perceived evidence. This takes the form of a prior distribution (the initial beliefs) and a likelihood function, where the priors are updated by multiplying them with the likelihood function, resulting in the posterior distribution.

Based on the famous Thomson Sampling (Thompson, 1933), a line of research proposes using the posterior of the Q-function to update parameters (Strens, 2001). In each episode, Posterior Sampling In Reinforcement Learning samples new parameters from the posterior, uses them for the rest of the episode, and acts optimally according to the policy specified by them.

Inferring the posterior is generally not feasible; to enjoy the different benefits that the posterior offers, algorithms need to rely on approximations of it. Making these approximations is in itself

a challenging task. Two different approaches are commonly used: Variational Inference, where a model is used to represent the posterior, and Markov Chain Monte Carlo methods, where samples are approximated directly from the posterior (Van Der Vaart et al., 2024). Each of these approaches comes with added challenges, and in some cases, the theoretical guarantees they offer are based on assumptions that can not be met in practice.

3.4 Langevin Monte Carlo

Langevin Monte Carlo (LMC) is a Markov Chain Monte Carlo (MCMC) method. MCMC methods are computational algorithms designed to sample from complex probability distributions, especially in situations where analytical solutions are infeasible. LMC belongs to this family of algorithms. The iterative update rule for LMC is given by:

$$X_{t+1} = X_t - \eta_t \nabla L(X_t) + \sqrt{2\eta_t}\xi_t \tag{3}$$

where η_t is the step size at time step t, $\nabla L(X_t)$ is the gradient of the value function; which in RL typically will be the negated loss function, and ξ_t, which is drawn from a multivariate Gaussian distribution in each time step t. This process will generate a Markov Chain that converges to a distribution $\propto exp(\nabla(L))$ under certain assumptions (Roberts and Tweedie, 1996; Bakry et al., 2014).

4 LMC-LSVI and Adam LMCDQN

In this section, we aim to give the reader a thorough understanding of the analysed algorithms. First, we delve into the mechanics of LMC-LSVI, and then, building on top of some of the concepts from LMC-LSVI, we explain how Adam LMCDQN works. For the full pseudocode of the algorithms, we refer the reader to Appendix A.

4.1 LMC-LSVI

The LMC-LSVI (Langevin Monte Carlo Least-Squares Value Iteration) algorithm is an RL method designed to improve exploration by directly sampling from the posterior distribution of the Q-function using LMC. LMC-LSVI does not rely on approximations of the posterior, which is a key component of the algorithm, as Gaussian distributions are not great surrogates in many cases, in addition to the difficulty of estimating the mean and variance.

In each episode, the algorithm updates the Q-function parameters for every time step of the episode by running several iterations of a noisy gradient descent. More concretely, the iterative parameter update is given by:

$$w_h^{k,j} = w_h^{k,j-1} - \eta_k \nabla \widetilde{L}_h^k(w_h^{k,j-1}) + \sqrt{2\eta_k \beta_k^{-1}}\epsilon_h^{k,j} \tag{4}$$

Where the index k represents the current episode training process, and each episode corresponds to a complete trajectory from an initial state to a terminal state in the environment. The index h represents the time step within an episode, corresponding to different decision points along the trajectory where the agent must choose actions. The index j denotes the gradient update step within the current episode k, as the algorithm performs a determined number J of parameter updates during each time step to better estimate the Q-function.

While being very similar to Equation 3, there are some differences worth noting. First, the inverse temperature β_k^{-1} term is introduced. This allows control of the scale of the injected noise, which can play a key role in exploring or exploiting. It is important to realise that higher values of β_k^{-1} lead to a smaller scale of noise. Then, $\tilde{L}_h^k$ represents the estimate of L_h^k made with a batch of data.

Each noisy gradient step consists of a standard gradient descent move on a regularised temporal difference error, plus an added Gaussian noise term scaled by a factor named inverse temperature.

Under the linear MDP setting (Jin et al., 2020), this algorithm has been proven to give a regret of $\widetilde{\mathcal{O}}(d^{3/2}H3/2\sqrt{T})$ (Ishfaq et al., 2024), where d is the feature dimension, H is the planning horizon, and T is the total number of steps.

4.2 Adam LMCDQN

Adam LMCDQN (Adam Langevin Monte Carlo Deep Q Network) builds on top of the theoretical guarantees of LMC-LSVI and combines it with Adam SGLD (ASGLD) (Kim et al., 2020) by replacing the LMC step with ASGLD when estimating the posterior. This combination leverages the adaptive learning rates from Adam and its momentum to stabilise training while maintaining exploration benefits from LMC.

$$w_h^{k,j} = w_h^{k,j-1} - \eta_k \Big(\nabla \tilde{L}_h^k(w_h^{k,j-1}) + a \text{ momentum term} + \sqrt{2\eta_k \beta_k^{-1}} \epsilon_h^{k,j} \tag{5}$$

This equation is very similar to Equation 3 but it adds the extra term a momentum term, where a is the bias factor and determines how much of the Adam momentum is used in the parameters optimisation. For a detailed explanation, we refer the reader to (Kingma and Ba, 2017), and to (Ishfaq et al., 2024), for the concrete adaptation for reinforcement learning.

It is important to note in Equation 5 that setting the bias factor $a = 0$ results in the update from the LMC-LSVI algorithm, and that these updates are done J_k times with the same batch of data for the estimation of $\nabla \widetilde{L}_k^k$.

5 Experimental Setup

This work aims to give a better understanding of the LMC-LSVI and Adam LMCDQN algorithms. We analyse the robustness of these algorithms by conducting empirical experiments with different hyperparameters. In this section, we introduce the different components of the conducted experiments. We first introduce the analysed environments in 5.1, then we explain the baseline used in our study in 5.2, and we then give an extensive overview of the parameters used in our study in 5.3. Finally, we explain the concrete details of our implementation in 5.4.

5.1 Environments

In this subsection, we present the different environments that will be later analysed to derive results. First, we present the chosen bandits, along with a short explanation of their design. Then we present the three chosen DRL environments. For an extensive explanation of the environments and their reward structures, we refer the reader to Appendix B.

The bandit settings analysed in this paper are the Gaussian Bandit (Lange and Sprekeler, 2022), the MNIST Bandit (Osband et al., 2020), and the Bernoulli Bandit (Wang et al., 2017). These multi-armed bandits exhibit different reward structures.

The **Gaussian Bandit** is a bandit with two arms, where one arm has a fixed reward of 0 and the other one has a reward drawn from a Gaussian distribution $N(\mu, \sigma_e)$, where σ_e remains constant across episodes and μ is sampled at the beginning of each episode from $N(-1, \sigma_p)$. The length of the episodes is fixed to 100 steps. We choose to use the default values provided by Gymnax: $\sigma_e = 0.1, \sigma_p = 1$.

The **MNIST Bandit** is a contextual bandit with 10 different arms, each representing a digit [0-9] and a 28x28 image representation of a digit as context. A reward of 1 is observed when correctly choosing the number, and otherwise, a reward of -1 is perceived.

The **Bernoulli Bandit** is a bandit where each arm k has a probability p_k of giving a reward of 1, otherwise it gives a reward of 0.

We have chosen three different DRL environments in this paper. First, the **Cart Pole Environment** (Barto et al., 1983), which has been chosen as it is an example of an exploitation-heavy environment. In this environment, a cart moves along a frictionless track, trying to balance a pole attached to it.

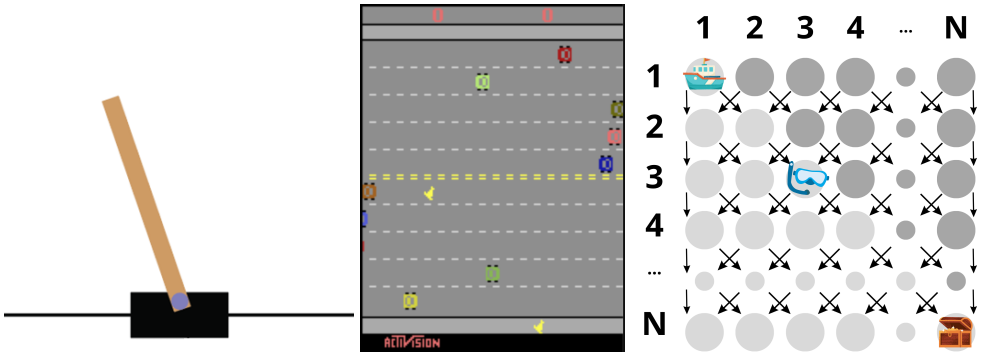

Figure 1: **Cart Pole Environment** (left), **Freeway Environment** (middle), **Deep Sea Environment** (right).

Second, the **Deep Sea Environment** (Osband et al., 2020), which is a very exploration-heavy environment where an agent has to choose to go towards the left or the right for $N - 1$ steps. We use the deterministic version without randomised actions of the environment.

Lastly, the third chosen environment is **Freeway**, the MinAtar version (Young and Tian, 2019). This environment represents the Freeway Atari game, part of the Atari 2600 game suite (Bellemare et al., 2013), but with simplified graphics. In the game, a chicken needs to cross a ten-lane street with cars crossing horizontally. This environment has a sparse reward structure, making it an exploration environment, though not as much as Deep Sea.

5.2 Baseline

In this study, we have chosen as a baseline the ε-greedy exploration strategy. This is considered to be one of the most naïve exploration strategies, yet it can show good performance in some environments. ε-greedy balances exploration and exploitation by taking a random action with probability ε and the best known action otherwise (probability of $1 - \varepsilon$). Typically, this ε value decays as the training progresses. We choose to linearly decay during a specified annealing time, starting at $\varepsilon = 1$ and ending at $\varepsilon = 0.05$.

5.3 Parameters Used

In this study, we focus on changing the hyperparameters that are specific to our algorithm, namely, inverse temperature, bias factor a, number of updates J_k and learning rate.

Since Reinforcement Learning algorithms are often highly sensitive to the hyperparameters used, we aim to give a detailed overview of the setup and the chosen hyperparameters in Appendix C. These will remain constant across experiments unless stated explicitly. We also do not change the network architecture, which we keep unmodified across experiments, runs and agents.

The chosen architecture for the Neural Network is defined as follows: First, there is a fully connected layer with 120 neurons, followed by a ReLU activation. Then, there is another fully connected layer with 84 neurons, again followed by a ReLU activation. Lastly, there is a last fully connected layer whose size is defined by the dimension of the action space (one neuron for each possible action).

5.4 Implementation Details

For the implementation of the algorithms, we used Python coupled with the JAX framework (Bradbury et al., 2018). JAX offers several key features, such as automatic differentiation and just-in-time compilation, that improve the performance of the agents.

DRL algorithms are bug-prone and difficult to implement from scratch, so we base our agents on the DQN algorithm implemented by PureJaxRL (Lu et al., 2022). Slight modifications have been made to the algorithm related to seed handling. We refer to our GitHub repository for concrete

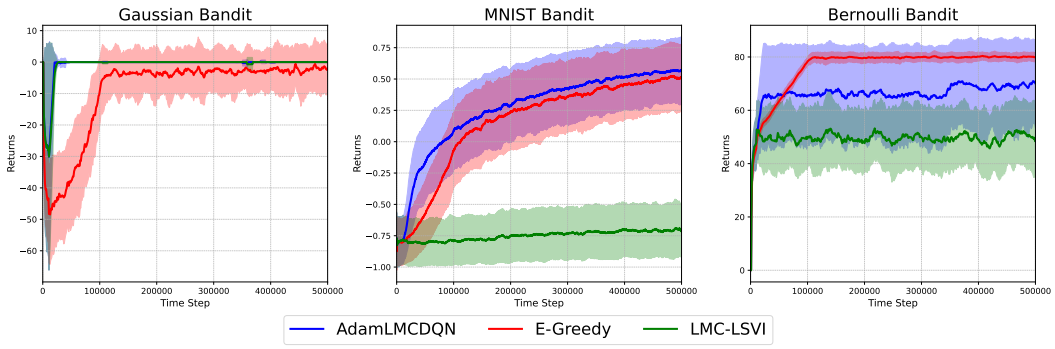

Figure 2: **Bandit Returns** for LMC-LSVI, Adam LMCDQN and ε-greedy. Environments, from left to right, Gaussian Bandit, MNIST Bandit and Bernoulli Bandit. All the lines represent the mean return over 10 runs, while the shaded area is the standard deviation. LMC-LSVI and Adam LMCDQN with an inverse temperature of 10^8.

implementation details. For the implementation of the environments, we chose the implementations provided by Gymnax (Lange, 2022).

6 Results

In this section, we present the results necessary for answering the research questions presented in the introduction by showing empirical results in different environments. Note that it is mentioned in several cases that results come from N runs from base seed x; this means that seed x is split into N seeds, and then for each of those, a run is made.

In addition to the presented plots, we include the same results plotted over time steps in Appendix D.

First, in 6.1, we will present the results of the bandit setups proposed in 5.1. Then we show results from different experiments conducted in the different environments to answer what is the effect of the number of updates J in 6.2, what is the effect of the noise scale, Adam weight, and learning rate in 6.3, and finally, in 6.4, what is the most robust set of hyperparameters.

6.1 Bandit Setups

In this subsection, we present the performance of LMC-LSVI and Adam LMCDQN in bandit settings. Since bandits are generally simpler than DRL environments, we run ε-greedy with decay through the first 100,000 steps. We also use 10 seeds for the bandit results since it is computationally feasible.

In Figure 2 (Left), we see that LMC-LSVI, Adam LMCDQN, and ε-greedy perform very similarly in the **Gaussian Bandit**. LMC-LSVI and Adam LMCDQN converge very quickly to the optimal strategy, choosing the bandit with a fixed reward of 0, while ε-greedy has a bigger learning time due to the annealing time. ε-greedy also shows some noisy behaviour due to the minimum value of epsilon being 0.05.

In contrast, in Figure 2 (Middle) we observe how in the MNIST Bandit Adam LMCDQN and ε-greedy perform similarly while LMC-LSVI performs much worse. This might be due to the algorithm not being able to correctly capture the dimensionality of the context and learn from it.

Finally, in Figure 2 (Right) we observe how ε-greedy outperforms Adam LMCDQN, and performs much better than LMC-LSVI in the Bernoulli Bandit. When looking at the individual runs of Adam LMCDQN, we see that in some cases it converges to rewards of around 80, while in other runs it presents very noisy returns ($\pm$ 20, approximately) centred around 50, causing the standard deviation to be quite large.

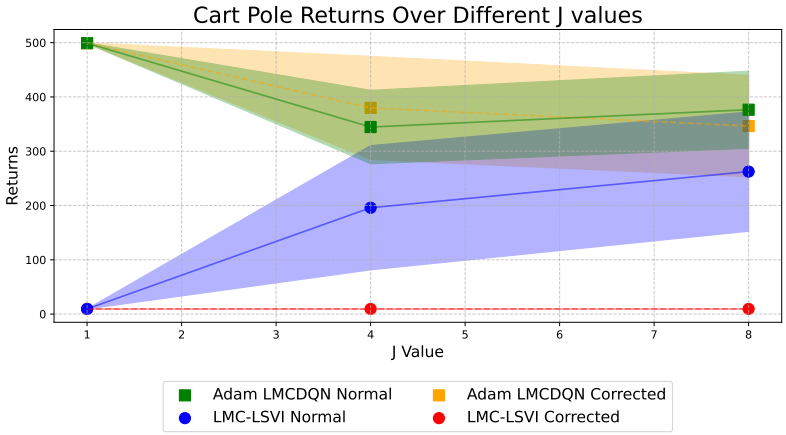

Figure 3: **Cart Pole Average Returns** of LMC-LSVI (circle) and Adam LMCDQN (square) over J values of 1, 4 and 8, with dashed lines representing runs with corrected learning rate, and solid lines without learning rate correction (normal). Points represent the mean of 5 seeds (base seed 42) with 500.000 step runs, while the shaded area is a single standard error. Hard lines connect points.

6.2 Effect of the number of updates J

In this subsection, we present the effect of increasing the number of updates J_k in the LMC-LSVI and Adam LMCDQN algorithms. For the sake of this study, we keep J_k constant across steps k, making $J = J_k$ for all k. The J parameter defines the number of times that each batch of data will be used to update the weights.

We consider three different cases, first $J = 1$, meaning we only perform one update for each sampled batch of data. Then we consider the cases $J > 1$ and $J > 1$, but adding a correction to the learning rate. Performing several updates with the same batch of data might lead to instabilities in training; therefore, we correct the learning rate in the following way: $lr' = lr/J$.

In Figure 3 we observe how in the CartPole environment LMC-LSVI does not perform good, and only manages to achieve some returns when using $J > 1$ without learning rate correction. We see that Adam LMCDQN performs quite well, achieving the maximum returns possible (500) when used with $J = 1$.

From these results, we can conclude that pairing Adam LMCDQN with $J > 1$, with or without learning rate correction, does not yield an improved performance in the Cart Pole environment. In contrast, LMC-LSVI manages to achieve relatively good performance with $J > 1$ without learning rate correction, suggesting that this algorithm might benefit from bigger learning rates.

In Figure 4 we see the effects in returns of changing J in the Freeway environment. Here, we only run Adam LMCDQN, and consider the case $J = 1$ and $J > 1$ without learning rate correction, as running these experiments is computationally expensive.

We see how increasing J degrades the performance of Adam LMCDQN in the environment, suggesting that $J = 1$ is the best choice for this environment. We also highlight that a higher J leads to more updates, making the algorithm computationally more expensive. This makes $J = 1$ not only the best performer but also the most computationally efficient.

Finally, in Figure 5, we observe LMC-LSVI achieves some results when used with $J = 1$ and $J > 1$ with learning rate correction. This does not happen for Adam LMCDQN, which achieves the best results when $J = 1$ and similar results with $J > 1$ without learning rate correction.

In this occasion, we observe that correcting the learning rate does slightly improve the performance of LMC-LSVI, though in contrast, it does not happen for Adam LMCDQN.

Based on the experimentation, we find that increasing J does not lead to a performance boost in the Adam LMCDQN algorithm; $J = 1$ achieves the best results across all the compared settings. For LMC-LSVI, we find substantial performance improvements in the CartPole environment when using

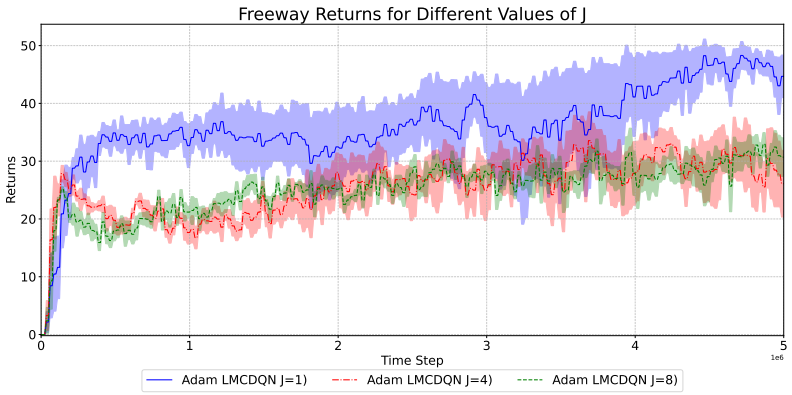

Figure 4: **Freeway Returns** for different values of J (without learning rate correction). We run Adam LMCDQN for 5 million steps with an inverse temperature of 10^8. Hard lines are the means over 5 seeds (base seed 42), and the shaded areas represent single standard errors. We smooth lines and standard errors over windows of 100.

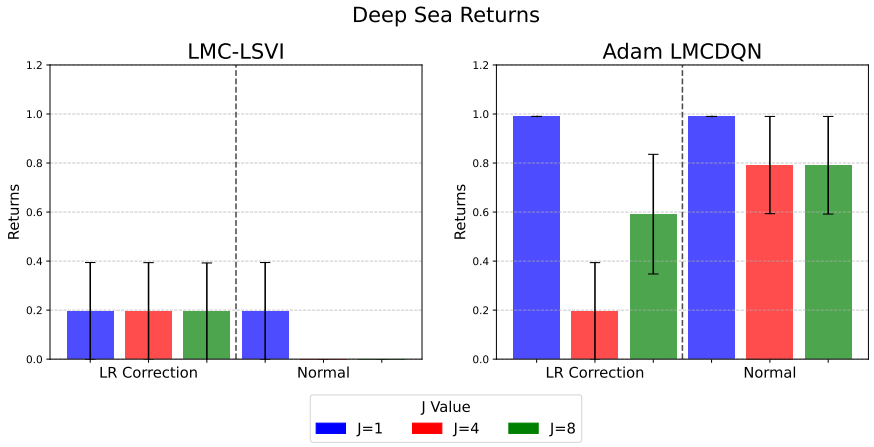

Figure 5: **Deep Sea (Size 20) Average Returns** for LMC-LSVI (left) and Adam LMCDQN (right) with an inverse temperature of 1000. Each box represents the mean return over 5 returns, and the whiskers are a single standard error. Each value of J has a different colour (1: blue, 4: red, and 8: green).

$J > 1$ without learning rate correction, though when looking at the other environments, we do not see any remarkable difference in performance that suggests that $J > 1$ is consistently better.

6.3 Effect of Noise Scale, Adam Weight and Learning Rate

In this subsection, we analyse the effect of changing the inverse temperature (β_k), which controls the scale of the injected noise, changing the bias factor (a), which controls the scale of the Adam momentum, and modifying the learning rate.

In the Deep Sea results from Figure 6, we observe that for this exploration-heavy environment, the choice of inverse temperature is very important, while the choice of a also affects the performance, but in a smaller way.

Having large inverse temperature values (greater than 10^8) leads to not being able to find rewards in most cases. We only manage to find some rewards when pairing these large temperatures with a high value for a as we can observe in the bottom row ($a = 5$).

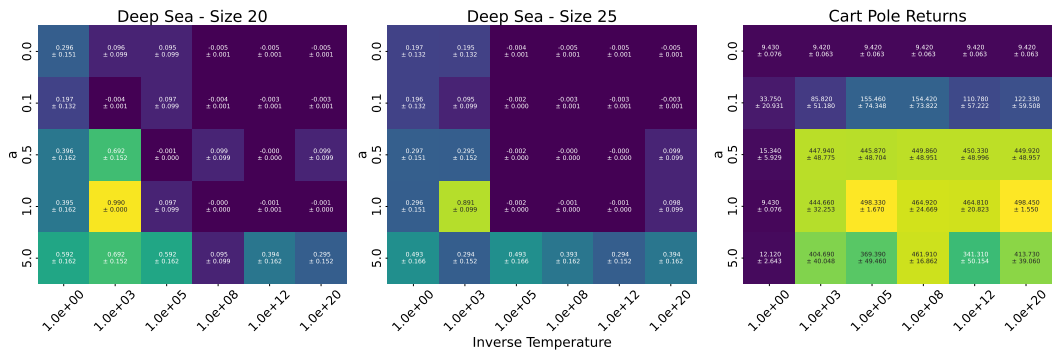

Figure 6: **Deep Sea Returns** and **Cart Pole Returns** for different values of a and inverse temperature. Deep Sea environments of size 20 and 25 (left and middle) and CartPole (right). Results presented as the mean from 10 seeds (5 from base seed 42, 5 from base seed 33) achieved after 500.000 steps $\pm$ a single standard error. Note that in the top row, $a = 0.0$, represents the LMC-LSVI algorithm.

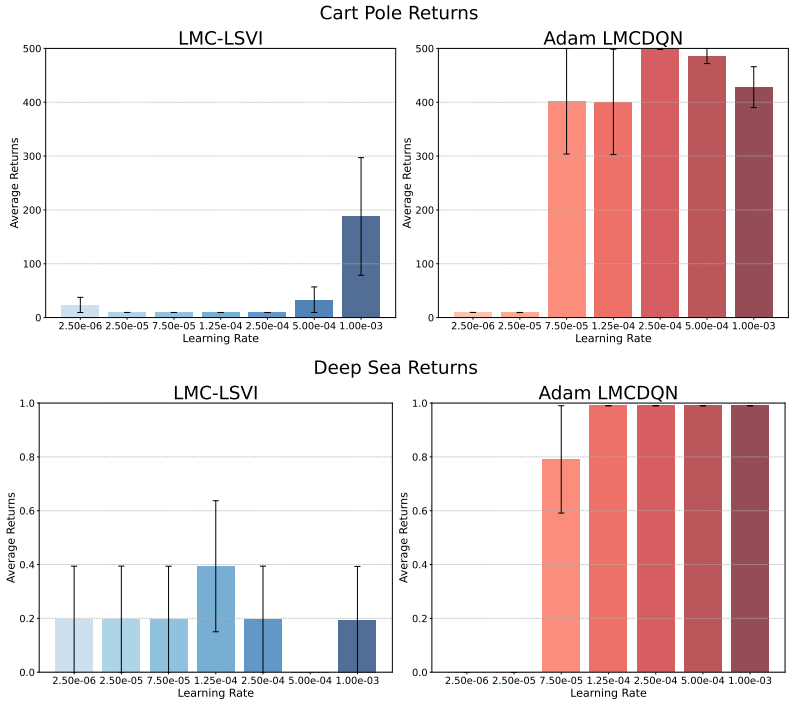

Figure 7: **Cart Pole Returns** (above) and **Deep Sea (Size 20) Returns** (below) for different learning rates, results presented as the average over 5 seeds (base seed 42) with the whiskers being a single standard error. All the runs ran for 500.000 steps with an inverse temperature of 10^8 for Cart Pole and of 10^4 for Deep Sea.

For a values, we observe that $a = 1$ gives the best performance, while $a = 0.5$ and $a = 5$ also give similar results. For a values lower than 0.5 and including LMC-LSVI, we do not observe any relevant returns. Something similar happens with Cart Pole, suggesting that low a values can lead to very low performances.

In contrast, the CartPole returns show that for this exploitation-heavy environment, the inverse temperature is not as relevant. We do not observe big differences with $a = 0.5$ and $a = 1$ for the chosen inverse temperature values (ranging from 10^3 to 10^{20}).

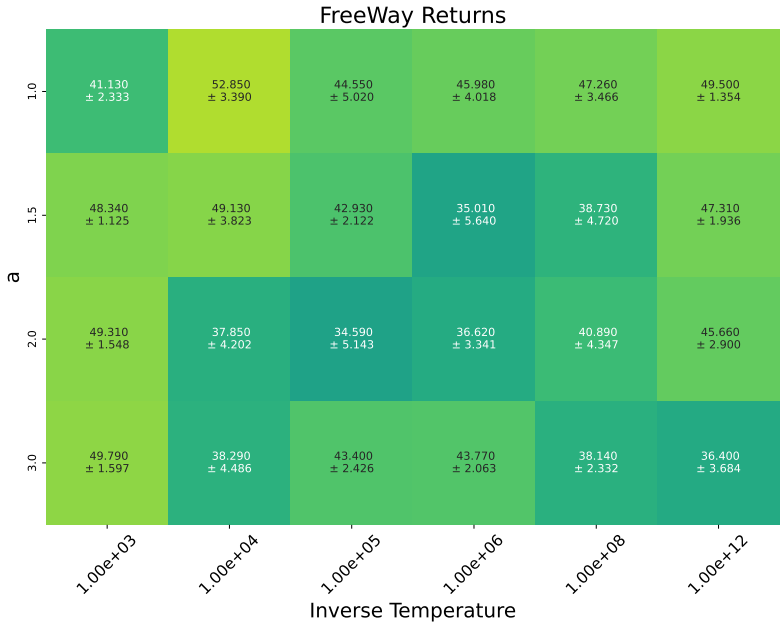

Figure 8: **Freeway Returns** for different values of a and inverse temperature. Every cell represents the average over 10 seeds (5 from base seed 42 and 5 from base seed 33) $\pm$ a single standard error. Each run with 5×10^6 steps.

Figure 7 shows that when changing the **learning rate** for LMC-LSVI, bigger values (10^{-3}) can lead to the algorithm achieving some results in the Cart Pole environment. This is in line with the findings of 6.2. In contrast, the algorithm does not exhibit a lot of change in the Deep Sea environment. We find Adam LMCDQN to be quite robust against different learning rate values, showing that the Adam LMCDQN algorithm carries some of the nice properties that the Adam optimiser offers. We also observe that, contrary to what the theory behind LMC suggests, smaller learning rates do not lead to better empirical results.

6.4 Most Robust Set of Hyperparameters

In this subsection, we build upon the results of the other subsections in order to find hyperparameters that yield good performance across environments.

First, we find that Adam LMCDQN outperforms LMC-LSVI in the analysed environments, except for the Gaussian Bandit, where they exhibit a very similar performance. For this reason, from this point onwards, we focus on Adam LMCDQN. Secondly, in 6.2 we find that $J = 1$ gives the best results, and that increasing it does not improve the performance even when correcting the learning rate. Lastly, in 6.3 we see how, while the choice of a good set of hyperparameters is environment-dependent, there is a range of hyperparameters that give similar performance to the best set. We observed that the algorithm was robust against different learning rates; we therefore chose to continue with 2.5×10^{-4}.

We propose the following set of hyperparameters as a robust set:

Parameter	Values
Inverse Temperature	10^3, 10^4, 10^5, 10^6, 10^8, 10^{12}
a	1, 1.5, 2, 3

Table 1: Proposed set of robust hyperparameters.

Since most of our hyperparameter experimentation has been derived from the Deep Sea and Cart Pole environments, we test the proposed set of hyperparameters in the Freeway environment. Figure 8 shows the performance of the selected set of hyperparameters in the Freeway environment. For

reference, ε-greedy has returns of 20.160 ± 6.211, ran across 5 seeds with an annealing time of 2×10^6 and a single standard error. We observe that there is a big gap between all the results from the proposed set of hyperparameters and the baseline, showing that this set of hyperparameters is robust in this environment.

7 Responsible Research

In this section, we reflect on the reproducibility and ethical implications of the conducted research. In this research, no human or sensitive data have been used, mitigating possible biases and reducing the direct ethical implications.

During this study, several measures have been taken to ensure reproducibility. First, we give an in-depth explanation of the components of our algorithm in this paper, and we also publish the concrete implementation as we make our code publicly available[1]. This allows the reader to check concrete details and reuse the implementation for the reproduction of the results. Additionally, we detail all the hyperparameters used as well as the network architecture used. Second, we put special care into seed handling, ensuring that the results received from our code are consistent and reproducible across runs. We also include logic in our code to split a single seed into as many as requested by the user, making the process of reproducing results straightforward.

Of central importance in modern research is the use of AI. In this research, we leverage AI tools for spell checking in the process of writing this paper. We also utilise AI to efficiently navigate documentation of the different libraries used, for example, JAX or matplotlib.

We also highlight the limitation in computing power. This has been a key factor in the overall research progress, limiting the feasible number of combinations of hyperparameters, seeds and length of experiments that can be run.

8 Conclusion and Future Work

In this paper, we set out to empirically research the robustness and sensitivity of the LMC-LSVI and Adam LMCDQN algorithms. We revisit the hyperparameters proposed in the original paper and provide experimentation in three different environments.

We researched the application of LMC-LSVI and Adam LMCDQN to bandit settings, with results showing that Adam LMCDQN generalises well to such settings even though in some cases it performs worse than ε-greedy. LMC-LSVI does not exhibit very good performance in most cases. We also provided empirical results for different hyperparameters. In our experiments, Adam LMCDQN consistently outperforms LMC-LSVI. For Adam LMCDQN we find $J = 1$ to give the best results, for inverse temperature (β_k) we find the range of 10^3 to 10^{12} to be the most robust across environments, with exploration heavy environments benefiting from lower ranges (10^3 to 10^8) and exploration heavy environments exhibiting good results in a wider range of values (10^3 to 10^{20}). We also show the robustness of Adam LMCDQN against different learning rates, with values ranging from 7.5×10^{-5} to 1×10^{-3} exhibiting very similar returns, we recommend 2.5×10^{-4}.

Following this study, there are several interesting lines of research. First, it would be interesting to research the application of Adam LMCDQN to real-world environments, for example, autonomous driving environments or financial environments. Second, experimenting with whether the application of other Markov Chain Monte Carlo methods, such as Hamiltonian Monte Carlo or Gibbs sampling, to Deep Reinforcement Learning can also yield good results. Finally, it would be interesting to bootstrap the LMC-LSVI and Adam LMCDQN algorithms and analyse the difference in performance with the results presented in this paper.

[1] Our code is available at https://github.com/PabloHendriks/AdamLMCDQN

References

Azizzadenesheli, K. and Anandkumar, A. (2019). Efficient exploration through bayesian deep q-networks.

Bakry, D., Gentil, I., and Ledoux, M. (2014). *Analysis and Geometry of Markov Diffusion Operators*, volume 348. Springer.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1957). A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684.

Benatan, M. and Pyzer-Knapp, E. O. (2019). Fully bayesian recurrent neural networks for safe reinforcement learning.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019). Exploration by random network distillation. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

Dwaracherla, V. and Roy, B. V. (2021). Langevin dqn.

Fan, J., Wang, Z., Xie, Y., and Yang, Z. (2020). A theoretical analysis of deep q-learning.

Fortunato, M., Azizzadenesheli, K., Tang, Y., et al. (2018). Noisy networks for exploration. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*.

Ishfaq, H., Lan, Q., Xu, P., Mahmood, A. R., Precup, D., Anandkumar, A., and Azizzadenesheli, K. (2024). Provable and practical: Efficient exploration in reinforcement learning via langevin monte carlo. In *International Conference on Learning Representations*.

Jin, C., Yang, Z., Wang, Z., and Jordan, M. I. (2020). Provably efficient reinforcement learning with linear function approximation. In Abernethy, J. and Agarwal, S., editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 2137–2143. PMLR.

Kim, S., Song, Q., and Liang, F. (2020). Stochastic gradient langevin dynamics algorithms with adaptive drifts.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Lange, R. T. (2022). gymnax: A JAX-based reinforcement learning environment library.

Lange, R. T. and Sprekeler, H. (2022). Learning not to learn: Nature versus nurture in silico.

Liu, R. and Zou, J. (2017). The effects of memory replay in reinforcement learning.

Lu, C., Kuba, J., Letcher, A., Metz, L., Schroeder de Witt, C., and Foerster, J. (2022). Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Osband, I., Aslanides, J., and Cassirer, A. (2018). Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*.

Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. (2016a). Deep exploration via bootstrapped dqn.

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvari, C., Singh, S., Roy, B. V., Sutton, R., Silver, D., and Hasselt, H. V. (2020). Behaviour suite for reinforcement learning.

Osband, I., Roy, B. V., and Wen, Z. (2016b). Generalization and exploration via randomized value functions.

Osband, I., Russo, D., and Roy, B. V. (2013). (more) efficient reinforcement learning via posterior sampling.

Roberts, G. O. and Tweedie, R. L. (1996). Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 2:341–363.

Russo, D. (2019). Worst-case regret bounds for exploration via randomized value functions.

Strens, M. (2001). A bayesian framework for reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning*.

Sutton, R. S. and Barto, A. (2020). *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts London, England, second edition edition.

Tang, H., Houthuijzen, R., Flet-Berliac, Y., et al. (2017). #exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems 30 (NIPS)*.

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

Van Der Vaart, P. R., Yorke-Smith, N., and Spaan, M. T. J. (2024). Bayesian ensembles for exploration in deep q-learning. *Proc. of the Adaptive and Learning Agents Workshop (ALA 2024)*.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2017). Learning to reinforcement learn.

Watkins, C. (1989). Learning from delayed rewards. pages 44–54.

Young, K. and Tian, T. (2019). Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments.

A Pseudocode for LMC-LSVI and Adam LMCDQN

In this appendix, we present the full pseudocode for the Langevin Monte Carlo Least Squares Value Iteration (LMC-LSVI) and Adam Langevin Deep Q Network (Adam LMCDQN) algorithms Ishfaq et al. (2024).

Algorithm 1 LMC-LSVI Ishfaq et al. (2024)

Require: Step sizes $\{\eta_k > 0\}_{k\geq 1}$, inverse temperature $\{\beta_k\}_{k\geq 1}$, loss function $L_k(w)$

Initialize $w_h^{1,0} = 0$ for $h \in [H]$, $J_0 = 0$
for episode $k = 1, 2, \ldots, K$ **do**
  Receive the initial state s_1^k
  for step $h = H, H-1, \ldots, 1$ **do**
    $w_{h,0}^k \leftarrow w_{h,J_{k-1}}^{k-1}$
    for $j = 1, \ldots, J_k$ **do**
      $\epsilon_h^{k,j} \sim \mathcal{N}(0, I)$
      $w_{h,j}^k \leftarrow w_{h,j-1}^k - \eta_k \nabla L_k^h(w_{h,j-1}^k) + \sqrt{2\eta_k \beta_k^{-1}}\epsilon_h^{k,j}$
    end for
    $Q_h^k(\cdot) \leftarrow \min(Q(w_{h,J_k}^k, \phi(\cdot,\cdot)), H - h + 1)^+$
    $V_h^k(\cdot) \leftarrow \max_a Q_h^k(\cdot, a)$
  end for
  for $h = 1, 2, \ldots, H$ **do**
    Take action $a_h^k \leftarrow \arg\max_{a\in\mathcal{A}} Q_h^k(s_h^k, a)$, observe reward $r_h^k(s_h^k, a_h^k)$ and next state s_{h+1}^k
  end for
end for

Algorithm 2 Adam LMCDQN Ishfaq et al. (2024)

Input: step sizes $\{\eta_k > 0\}_{k\geq 1}$, inverse temperature $\{\beta_k\}_{k\geq 1}$, smoothing factors α_1 and α_2, bias factor a, loss function $L_k(w)$.
Initialize $w_h^{1,0}$ from appropriate distribution for $h \in [H]$, $J_0 = 0$, $m_h^{1,0} = 0$ and $v_h^{1,0} = 0$ for $h \in [H]$ and $k \in [K]$.
for episode $k = 1, 2, \ldots, K$ **do**
  Receive the initial state s_1^k.
  for step $h = H, H-1, \ldots, 1$ **do**
    $w_h^{k,0} \leftarrow w_h^{k-1,J_{k-1}}$, $m_h^{k,0} \leftarrow m_h^{k-1,J_{k-1}}$, $v_h^{k,0} \leftarrow v_h^{k-1,J_{k-1}}$
    for $j = 1, \ldots, J_k$ **do**
      $\epsilon_h^{k,j} \sim \mathcal{N}(0, I)$
      $w_h^{k,j-1} \leftarrow w_h^{k,j-1} - \eta_k(\nabla \tilde{L}_h^k(w_h^{k,j-1}) + a m_h^{k,j-1} \oslash \sqrt{v_h^{k,j-1} + \lambda 1}) + \sqrt{2\eta_k \beta_k^{-1}}\epsilon_h^{k,j}$
      $m_h^{k,j} = \alpha_1 m_h^{k,j-1} + (1-\alpha_1)\nabla \tilde{L}_h^k(w_h^{k,j-1})$
      $v_h^{k,j} = \alpha_2 v_h^{k,j-1} + (1-\alpha_2)\nabla \tilde{L}_h^k(w_h^{k,j-1}) \odot \nabla \tilde{L}_h^k(w_h^{k,j-1})$
    end for
    $Q_h^k(\cdot) \leftarrow Q(w_h^{k,J_k}, \phi(\cdot))$
    $V_h^k(\cdot) \leftarrow \max_{a\in\mathcal{A}} Q_h^k(\cdot, a)$
  end for
  for step $h = 1, 2, \ldots, H$ **do**
    Take action $a_h^k \leftarrow \arg\max_{a\in\mathcal{A}} Q_h^k(s_h^k, a)$, observe reward $r_h^k(s_h^k, a_h^k)$ and next state s_{h+1}^k.
  end for
end for

B Explanation of the Environments

B.1 Cart Pole

Barto et al. (1983) first proposed the Cart Pole Environment. In this environment, an agent is tasked with controlling a cart that moves along a frictionless, linear track. Attached to the centre of the cart is a pole that can pivot and fall left or right. The objective is that the agent maintains the pole balanced upright by applying discrete forces (left or right) while staying inside some predefined boundaries.

The reward structure of this environment is considered to be dense. The agent receives a positive reward (+1) at every time step that the pole stays in a specified angular range of inclination, and the agent stays inside the boundaries. After 500 time steps without termination, the game ends, making the maximum achievable reward 500. The continuous reward signal provides rich feedback to the agent, making it relatively easy to associate specific actions with outcomes.

The state space of this environment is continuous, with variables such as velocity, position, pole angle, ... Because of this, the state space is of intractable size, thus requiring that the agent generalises across similar states. The same does not apply to the action space, where there are only two possible actions.

In terms of exploration, Cart Pole is not an exploration-heavy environment. Once a strategy is found that works well, there is no benefit in trying a different action, as there are no possible bigger future rewards. Therefore, while exploration is necessary in the early stages of the game to discover the mechanics, after a good policy is found, trying new policies is penalised.

B.1.1 Deep Sea

The Deep Sea Environment is part of the bsuite benchmark suite Osband et al. (2020), it consists of an agent that needs to navigate a NxN grid-like world where the agent is initialized in the top-left corner, coordinate $(1, 1)$, and at each time step it has to decide between taking the down left action or the down right action. The episode ends once the agent has reached the bottom of the grid, which is after $N - 1$ actions.

The rewards of this environment are extremely sparse. These are structured such that for each move taken to the left diagonal, a reward of 0 is observed, and for each move to the right diagonal, a reward of $-0.01/N$ is given. Lastly, there is a big magnitude reward of 1 in the bottom right of the environment. The optimal policy is then to go right bottom in every action, as there is no other way of observing the big reward of 1, which will result in an episodic return of 0.99.

This reward structure makes it very hard for agents to discover the optimal policy; the apparent short-term best action is to go towards the left, as that way there is no negative reward observed, though to maximise returns, going right at every step yields the maximum reward.

This environment is a great example of the exploitation-exploration tradeoff, sacrificing short-term rewards in favour of exploring with the hope of finding bigger long-term rewards later in the episode. As the size N of the environment increases, the likelihood of discovering the optimal policy through random exploration decreases exponentially.

B.1.2 FreeWay MinAtar

Freeway is an Atari 2600 game Bellemare et al. (2013). In this game, the player controls a chicken whose objective is to cross a ten-lane road where there are cars crossing. If hit by a car, the chicken returns to the start of the road, and if the chicken reaches the end of the road, a point is added to the score.

In our study, we use the MinAtar Freeway implementation Young and Tian (2019). This is a simplified version of the game, where the input size is greatly reduced, allowing agents to focus on the mechanics of the game itself rather than on understanding the pixel representation of it. In this implementation, the agent only has three possible actions: forward, back and do nothing. After reaching the end, the speed and direction of the cars are randomised. The agent is restricted to only being able to move every three frames, while car speeds vary from moving every frame to once every 5 frames. The length of the episode is 2500 time steps, after which the reward of the episode is equal to the number of times the agent reached the other end of the road.

The reward structure of this environment is considered to be sparse. The only reward that is observed by the agent is when reaching the end of the road, which yields a $+1$ reward. When hit by a car, no negative reward is observed, and the agent is simply returned to the beginning of the road. This makes learning challenging, as agents do not receive a negative signal when hit by a car.

C Used Parameters

In this appendix, we include the detailed list of hyperparameters used.

Parameter	Value
Num Envs	10
Buffer Size	10000
Buffer Batch Size	128
Total Timesteps	5.0×10^5
Target Update Interval	500
Learning Rate	2.5×10^{-4}
Learning Starts	10000
Training Interval	10
Gamma	0.99
Tau	1.0
Seed	42
Num Seeds	5
eps	1.0×10^{-8}
ϵ-greedy Parameters	
Epsilon Start	1.0
Eplsion Finish	0.05
Epsilon Anneal Time	2.5×10^5
LR Linear Decay	true
LMC-LSVI and Adam LMCDQN Parameters	
alpha1	0.9
alpha2	0.999
Inverse Temperature	1.0×10^8
J	1
a	1

Table 2: Experiment Parameters

D Results Over Time Steps

In this appendix, we include the results plotted over time steps. We do not include the plots that were already plotted over time steps. We present all the plots with hard lines representing means over different seeds and a single standard error.

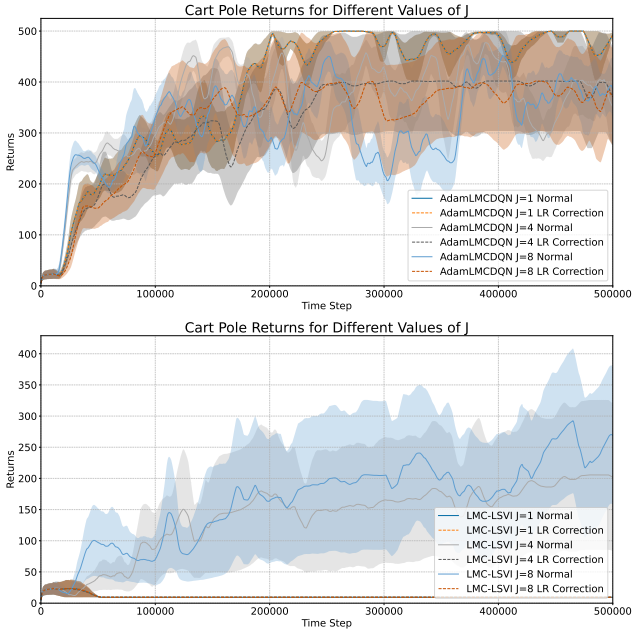

Figure 9: **Cart Pole Returns** for different values of J. Results averaged from 5 seeds and presented with a single standard error.

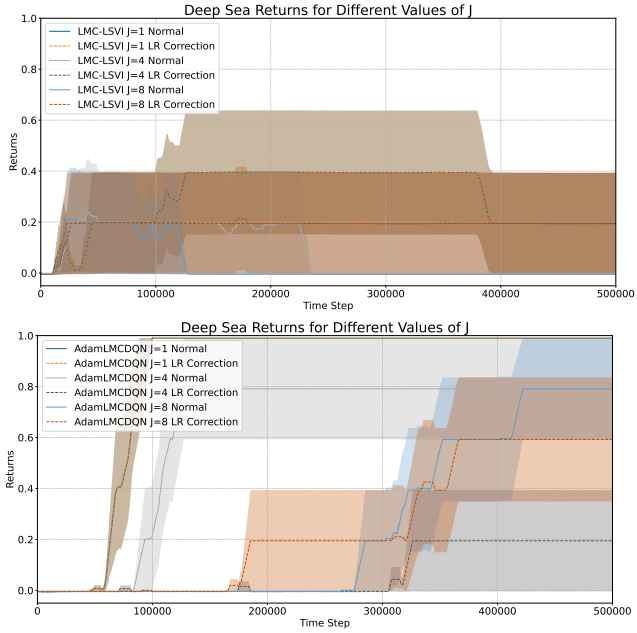

Figure 10: **Deep Sea Returns** of size 20 for different values of J. Results averaged from 5 different seeds and presented with a single standard error.

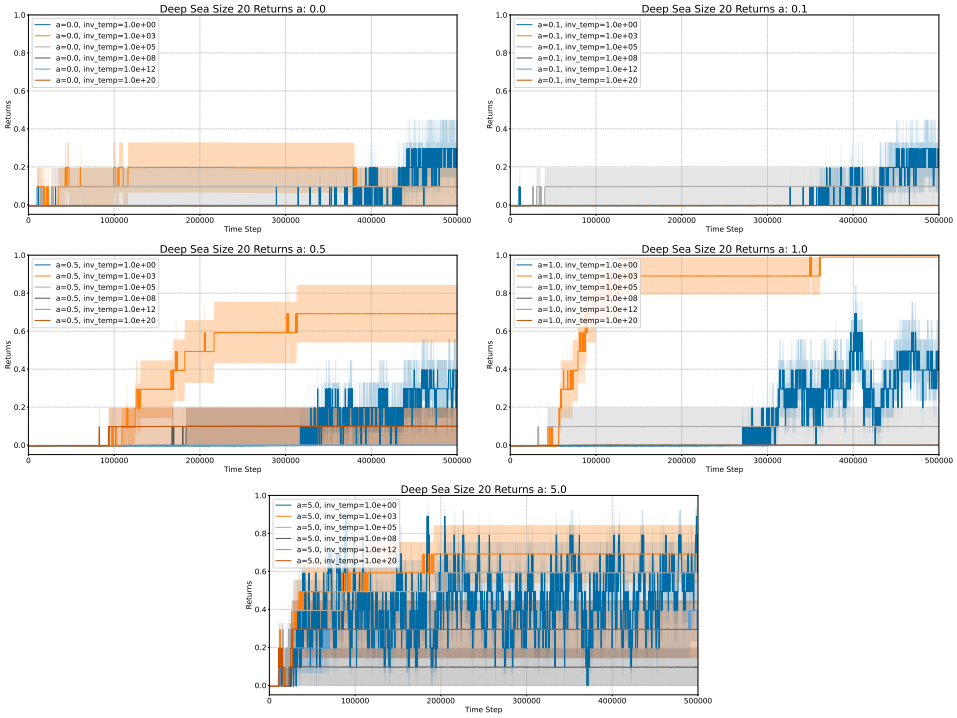

Figure 11: **Deep Sea Returns (Size 20)** for different values of a (one plot for each) and different inverse temperature values. Plots without windowing and shown as the average over 10 seeds and with a single standard error.

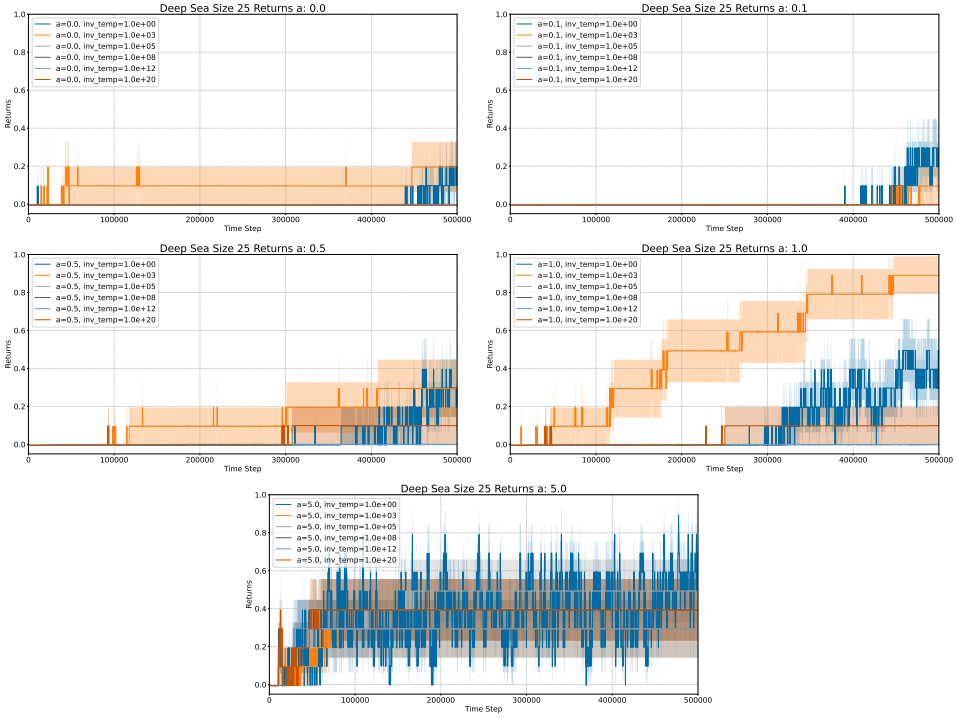

Figure 12: **Deep Sea Returns (Size 25)** for different values of a (one plot for each) and different inverse temperature values. Plots without windowing and shown as the average over 10 seeds and with a single standard error.

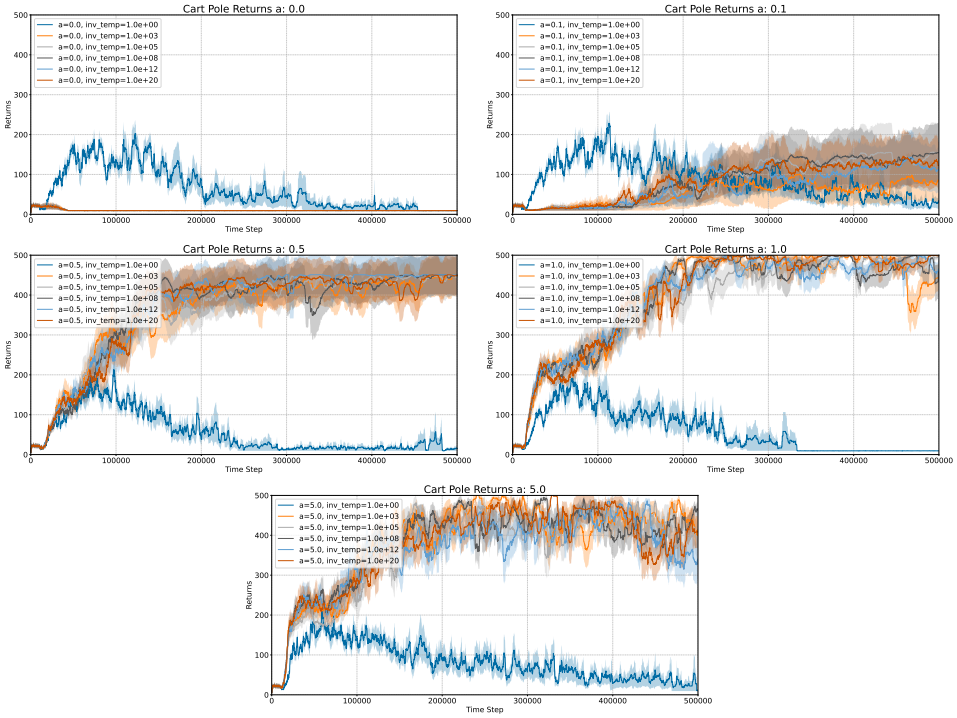

Figure 13: **Cart Pole Returns** (above) and **Deep Sea Size 20 Returns** (below) for different values of a (one for each plot) and different values of inverse temperature. Lines averaged from 10 seeds and presented with a single standard error.

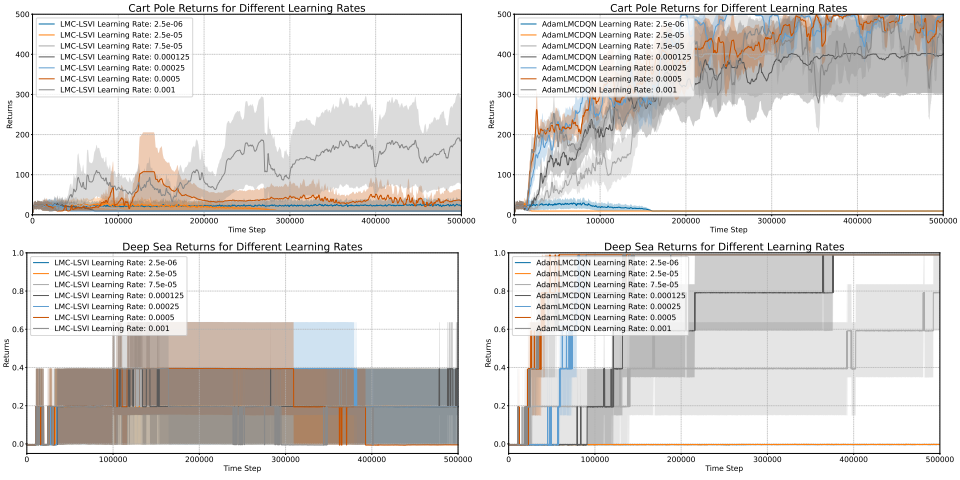

Figure 14: **Cart Pole Returns** for different learning rates. Hard lines represent averages over 5 seeds and shaded areas a single standard error.

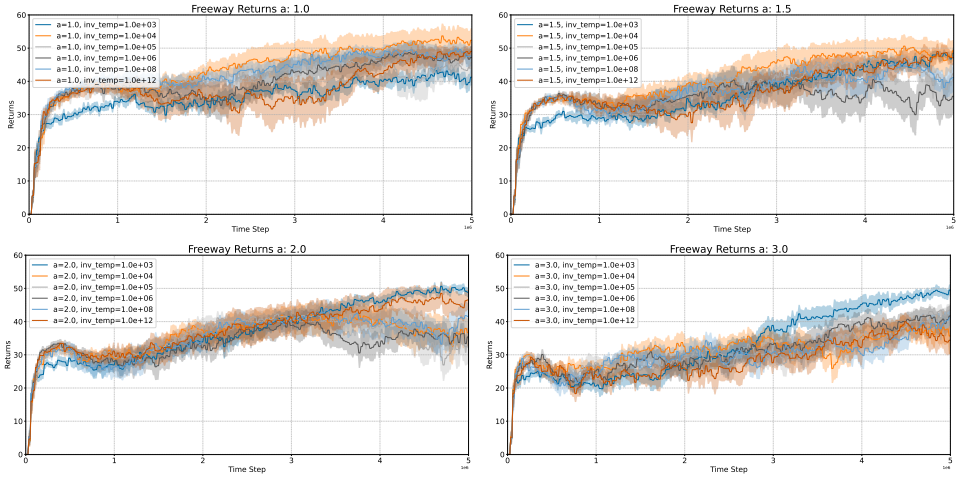

Figure 15: **FreeWay Returns** for different values of a (each value with its own plot) and different inverse temperatures. Hard lines represent means over 5 runs and shaded areas a single standard error.