Using NoisyNet to Improve Exploration in Contextual Bandit Settings

Sonny Ruff

Supervisor(s): Neil York-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: Sonny Ruff
Final project course: CSE3000 Research Project
Thesis committee: Neil York-Smith, Pascal van der Vaart, Matthijs Spaan

Abstract

Efficient exploration is a major issue in reinforcement learning, particularly in environments with sparse rewards. In these environments, traditional methods like ε-greedy fail to efficiently reach an optimal policy. A new method proposed by Fortunato, et al. [1] showed promise by improving efficiency on RL tasks such as Atari games, by driving exploration with learned perturbations of the network weights. Three different types of settings were investigated in order to test the robustness of a contextual bandit implemented with this proposed method: 1. ContextualBandit-v2; a bandit with multiple predefined functions mapping the 1-dimensional continuous input to the reward, 2. MNISTBandit-v0; a bandit rewarding correct identification of MNIST dataset images, and 3. NNBandit-v0; a bandit with the reward being determined by a neural network. Furthermore, non-stationary variants of environment 1. and 3. were tested. A slight variation in hyperparameter sensitivity between environments was observed and a generally optimal set was determined. Overall, NoisyNet-DQNs (Deep Q-Networks) achieved performance comparable to regular DQNs, though often slightly lower. In the high-dimensional stationary MNISTBandit-v0 environment, NoisyNet-DQN converged to an optimal policy slightly faster, at the cost of a larger variation in performance.

Keywords: Deep reinforcement learning, Exploration, Sparse reward problems, Contextual bandit settings

1 Introduction

Deep reinforcement learning (DRL) has achieved superhuman performance in tasks ranging from ATARI video games to complex robotic control. However, state-of-the-art methods often require billions of environment interactions before approaching human-level performance. Improving the efficiency of exploration could reduce this cost significantly, potentially enabling agents to match or even exceed human capabilities. Efficient exploration remains a central challenge in reinforcement learning, especially in environments with sparse rewards, where rewards are infrequent and agents struggle to learn the association between actions and outcomes [2]. Standard methods such as ε-greedy [3] and entropy regularization [4] introduce random perturbations to the agent's policy to encourage exploration. However, these methods typically lack the structure necessary to guide consistent, long-term behavioral patterns [1].

This study builds on the approach proposed by Fortunato, et al. which was published in 2018 [1], known as "NoisyNet", which will be referred to by that name throughout this paper. NoisyNet replaces traditional exploration heuristics, such as ε-greedy in DQN, with learned perturbations of the network weights, sampled from parametric noise distributions. This method results in the degree of exploration being state dependent and has shown to significantly improve the performance of agents playing a variety of Atari games. Atari games represent full RL settings involving sequential decision-making and delayed rewards. In a similar way, contextual bandits are simplified RL problems, consisting of a single-step environment with immediate rewards. NoisyNet did not investigate the effect of their method implemented in contextual bandit settings. These settings present an interesting opportunity, as they are common in real-world applications, such as online recommendation systems and personalized advertising [5].

To determine the effectiveness of NoisyNet's methods on contextual bandit settings, the following questions were asked:

1. To which hyperparameters is a NoisyNet-DQN sensitive, and are optimal values generally task-dependent?
2. On what classes of tasks does a NoisyNet-DQN significantly outperform or underperform a regular DQN?
3. What qualitative insights can be gained from visual evaluation metrics of the agent's learned behavior?

The structure of the following sections will be as outlined below: Section 2 details the studies background, first briefly formalizing and explaining a number of concepts used throughout the paper, and secondly detailing a number of related work. Section 3 explains the structure of the agent, neural network and the environments. Section 4 presents the experiment details and their results. Section 5 explains the considerations taken to ensure the integrity of the research. Section 6 discusses the presented results. And lastly, section 7 concludes the research and presents future research directions.

2 Background

2.1 Preliminaries

In this section, the basic concepts underlying this study are formally defined, starting with Markov decision processes, reinforcement learning and deep reinforcement learning, and lastly multi-armed and contextual bandit settings.

Markov Decision Processes (MDPs)

As described in Fortunato, et al. "MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future reward." [1]. They are defined by a tuple $\langle S, A, P, R, \gamma \rangle$. S being the set of states, A the set of actions the agent can take, P the transition probabilities, R the reward function, and $\gamma \in (0, 1]$ a discount factor. An important characteristic to note is that the set P holds the Markov property, which describes that any state only depends on its direct predecessor, formalized as:

$$P(s_i|s_{i-1}a_{i-1}s_{i-2}a_{i-2}...) = P(s_i|s_{i-1}a_{i-1})$$

Reinforcement Learning (RL)

Reinforcement learning is the third of the three main machine learning paradigms, alongside supervised and unsu-

pervised learning. While supervised learning aims to replicate behavior from examples, and unsupervised learning attempts to find hidden structures in data, reinforcement learning aims to maximize a reward signal [3]. This brings forth the exploration-exploitation dilemma; maximizing the reward by choosing the best action chosen in the past, while ensuring sufficient exploration to avoid missing better options.

To estimate the future rewards, reinforcement algorithms make use of value functions, evaluating the expected reward in a particular state. The state-value function $V^{\pi}(s)$ estimates the expected reward when starting from state s and following policy π:

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}[G_t|S_t = s] \tag{1}$$

This can be expanded to include the action taken, creating the action-value function (or q-value function). This estimates the expected reward when taking action a in state s and following policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}[G_t|S_t = s, A_t = a] \tag{2}$$

In order to apply these functions in machine learning, they can be rewritten in the form of loss functions:

$$L(\theta) = \mathbb{E}_{(S,A,P,R,\gamma)}[\frac{1}{n}\sum_{i=1}^{n}\underbrace{(\underbrace{(r + \gamma Q(s', a')}_{\text{temporal difference target}} - Q(s, a))^2}_{\text{temporal difference error}}] \tag{3}$$

Deep Reinforcement Learning (DRL)
While reinforcement learning utilises (q-)value function to update a table, deep reinforcement learning uses a deep neural network to approximate the optimal action-value function [1; 6]. An example of such an algorithm, and the basis of this study's code, is the Deep Q-Network (DQN) [6].

Multi-armed and Contextual Bandit Settings
A multi-armed bandit setting can be considered as a one-state MDP, with multiple transitions looping back to this one-state, each with a set reward. Each iteration, without prior knowledge of the rewards, the agent chooses one transition (or arm) according to its policy, and observes the associated reward. Since there is only a single state, the action-value function $Q(s, a)$ can be simplified to $Q(a)$, without dependence on state s. Consequently, without the existence of a previous state, the temporal difference target is reduced to $\widehat{Q} \leftarrow r$.

$$L(\theta) = \mathbb{E}_{(S,A,P,R)}[\frac{1}{n}\sum_{i=1}^{n}(r - Q(s, a))^2] \tag{4}$$

In practice, scenarios in which rewards are independent of context are rare. Contextual bandits are therefore often more appropriate for modeling such situations. One way to represent this is by reintroducing states, but calculating the q-value function without the transition between them.

2.2 Related Works

The following section outlines several prior studies related to this study.

Improvements in contextual bandit settings
Several other methods to improve exploration in contextual bandit settings, have been suggested. These include techniques like random network distillation [7], randomized prior functions [8], Monte Carlo dropout [9], and sampling-based methods such as Langevin Monte Carlo [10], though the effectiveness of these methods is under investigation.

NoisyNet in classic RL settings
NoisyNet suggests that the exploration efficiency of an RL algorithm could be improved by driving exploration using learned perturbations of the network weights. In traditional sparse reward RL settings, the difficulty of finding these rewards depends on navigating a specific sequence of states. In contrast, the contextual bandit setting lacks this dependency. Each state is independent and actions do not influence future states and rewards. Therefore, exploration in contextual bandit settings focuses on finding the optimal action in different contexts. As previously described, due to contextual bandit problems in essence being simplified RL problems, the techniques are relatively straightforward to carry over.

3 NoisyNet in contextual bandits settings

3.1 Implementation Details

This study is an application of exploration methods, originally developed for RL settings, to contextual bandit settings. Although the general structure of a DQN is used, it is important to clarify that the implementation is not a proper DQN, as no regular q-value algorithm is being applied. There is no target network and no learning from transitions between states, since each state is terminal and independent. This means that the loss calculation has been reduced to a simple calculation of the mean squared error between the observed reward and the prediction of the network.

The NoisyNet-DQN implementation from [11] was used as a starting point. Although no official implementation of the referenced work was available, multiple unofficial implementations were found online. However, these implementations varied slightly from the original pseudocode presented in the paper. The resampling of the noise is performed at different points in the algorithm, such as; every set number of training iterations [12], at the end of the loss calculation [13], or in one case, not at all [11].

Furthermore, no separate target and policy networks were used and the target network and its associated code were removed from the implementation.

3.2 Environments

Bandit settings require significantly less environment interaction to learn an effective policy compared to full RL problems like Atari games, enabling faster experimentation. As a result, a broader and more precise investigation could be conducted, including multiple experiments on different environments, and an investigation of the hyperparameter sensitivity for each.

In this study, three different contextual bandit settings were used; **1.** a bandit with multiple predefined functions mapping the 1-dimensional continuous input to the reward, **2.** a bandit rewarding correct identification of MNIST dataset images,

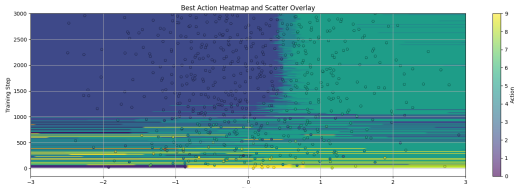


Figure 1: Decision plot of NoisyNet-DQN agent; seed 8796. Background color indicates the networks prediction of the optimal arm at the given training step, and overlaid the input values colored by arm chosen.

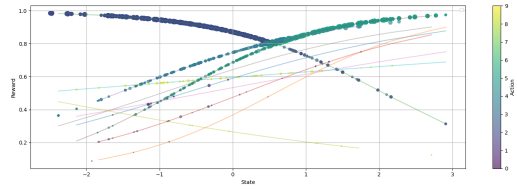


Figure 2: Reward plot of NoisyNet-DQN agent; seed 8796. Dots represent the reward observed in specific states, depending on the action taken. Larger dots indicate actions taken at later training steps. To clarify the possible rewards, the reward functions are plotted in the background. Note that these plots do not span the entire input range, as they are build up from randomly sampled data.

and **3.** a bandit with the reward being determined by a neural network.

Simple contextual bandit

The simplest bandit setting was based on the bandits of [14], with changes to the input range. Instead of using discrete states, the input state was made a continuous range, being sampled from a normal distribution with $\mu = 0$ and $\sigma = 1$. Each arm was defined as a function of the input state, as seen in figure 2. Upon initialization, every arm was assigned a random intercept and slope value.

To further test non-stationary behavior, these parameters were changed by a small random offset. Whereas this setting provides a non-zero reward for almost all input states and actions, the following sections will describe sparse-reward settings.

MNIST bandit

To provide a higher-dimensional sparse-reward bandit, the MNIST bandit from Google DeepMind's BSUITE [15] was partially used. No convolutional layer was used and the image data was directly passed through a fully connected layer. Although the network followed the structure of a classifier, it was only used to predict rewards rather than perform classification. As the MNIST dataset consists of handwritten digits [16], a reward of 1 was given only when the prediction correctly identified the digit in the input image. Any incorrect predictions resulted in a reward of 0.

No non-stationary behavior was implemented for this bandit, as there was no method of gradually shifting the optimal reward.

Neural network bandit

Lastly, the neural network bandit used a neural network, with random weights and biases, to determine the reward. This network remained the same throughout training and was not updated via backpropagation. The input to the reward network was the current state, while the number of actions available corresponded to the number of output nodes of the reward network. The reward received for a chosen action was simply the output value of the corresponding node. There was a consideration to be made regarding the input size of the reward network. At an input dimension of 1, the reward structure closely matched the reward structure of the simple contextual bandit. At much higher ($>$100) input dimensions, the training time grew too long, and convergence to an optimal policy slowed down drastically. To strike a balance between these factors, an input dimension of 10 was chosen.

Non-stationary behavior could be achieved relatively easily by resampling the weight and bias values of the network. This would however not result in interesting results, essentially switching to an entirely new reward function each time. Instead, a more gradual change was implemented. Given the current set of weights and biases N_i of the reward network, the resampled set of weights and biases $N^\star$, the values of the new reward network can be calculated by adding a fraction of the difference between N_i and $N^\star$, to N_i:

$$N_{i+1} = N_i + \frac{N^\star - N_i}{p}$$

Here, p determines the size of the change, with smaller values resulting in a smoother transition. This method allowed for non-stationary behaviour, without completely discarding previously learned behavior.

4 Experiments

So far this paper has mostly covered explanations of topics and descriptions of the study. The following section will discuss the concrete experiments performed.

For each experiment, a number of metrics were being tracked, to compare results, evaluate performance and eventually draw conclusions. Some metrics were tracked at every training step, some were aggregated from those metrics, and others were sampled every N training steps. Metrics tracked during training were a *smoothed reward*, *loss* and *regret*. Summary metrics that were used were the *final loss*, *mean regret*, *final regret* and *final score*.

For environments with a 1-dimensional input range, for the purpose of visualization of the behavior of the agent, another sample was taken. Every 10 training steps, the predicted optimal action was sampled of 100 uniformly distributed input states, as plotted in figure 1. From this, conclusions about the stability and the decision boundaries of the agent's policy could be drawn.

To determine which hyperparameters NoisyNet-DQN was most sensitive to, the hyperparameter sweeps facilitated by Weights and Biases, were used. For each sweep, a goal metric had to be chosen; minimizing *mean regret* was chosen for the purposes of this study. This choice was made due to this metric, besides indicating performance, also shows learning efficiency and whether the final policy is optimal. Their system rates the hyperparameters explored during the sweep with two metrics: *correlation* and *importance*. *Correlation* is the linear

correlation between the hyperparameter and the chosen goal metric, but fails to capture second order interactions. *Importance* improves on this, by training a random forest with the hyperparameters as inputs and the chosen metric as the target output, and reporting the feature importance values [17]. This sweep were performed with the hyperparameters listed in table 1, on each environment, with NoisyNet and without NoisyNet.

Hyperparameter	Values
method	random
noisy_layer_distr_type	[uniform, normal]
noisy_output_layer	[true, false]
noisy_reward	[true, false]
hidden_layer_size	[4, 8, 16, 20, 24, 40, 80]
noisy_layer_init_std	[1e-4, 1e-3, ..., 1e6]
# of samples	100

Table 1: Stationary environment sweep hyperparameters

The parallel coordinate plots showing the results of the hyperparameter sweeps, and the most influential hyperparameter per environment for the NoisyNet and Non-NoisyNet agent are included in appendix A.

The noisy_layer_distr_type hyperparameter determined the distribution type (either normal or uniform) from which the noisy linear layer sampled the noise. It was omitted from further sweeps, due to its low importance. In contrast, noisy_reward hyperparameter, which offsets the final reward by a random amount, showed very high influence on the agents behavior. Given this effect and the non-deterministic nature this gave the environment, it was decided to also omit this hyperparameter.

Next, a number of hyperparameters were investigated in greater depth to determine their influence on performance.

The most important hyperparameter across all experiments, was hidden_layer_size. As the name implies, the hyperparameter hidden_layer_size determines the size of the hidden layer of the agents' neural network. A small hidden layer struggles to properly learn the patterns in the input, especially in the high dimensional input space of MNISTBandit-v0. Although MNISTBandit-v0 showed erratic results, at a hidden_layer_size of 40 and higher, the mean_regret observed remained relatively constant in all three environments, as seen in figure 3.

The second most important hyperparameter, only applicable to the NoisyNet agents, was the noisy_layer_init_std. It determined the initial value of σ_0 used by the noisy linear layer, driving the exploration rate. As shown in figure 4, variance in mean_regret increases significantly in ContextualBandit-v2 and NNBandit-v0 when the parameter drops below 0.3. A value of 0.4 was selected as a balance between low mean regret and low variance.

A separate grid sweep, with the hyperparameters listed in table 2, was performed to determine optimal values for the replay buffer maximum memory size (memory_size) and sampling batch size (batch_size). These hyperparameters are particularly important in non-stationary environments, as they determine whether the agent learns from outdated observations.

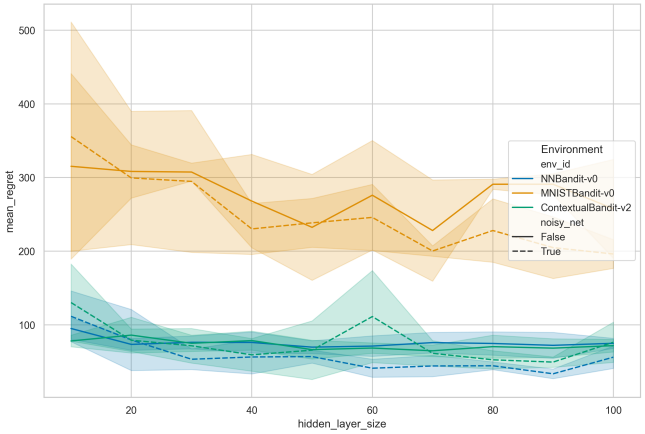


Figure 3: Mean regret ± 1 std. for varying hidden_layer_size values (step size 10) for NoisyNet and Non-NoisyNet agents, tested on seeds 0 to 4 for all three stationary environments.

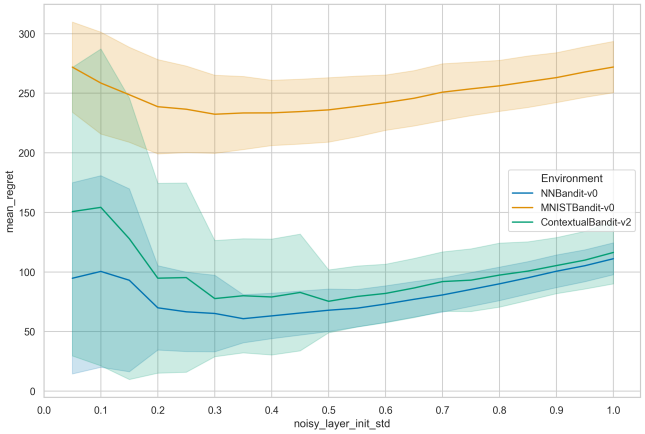


Figure 4: Mean regret ± 1 std. for varying noisy_layer_init_std values (step size 0.05) for NoisyNet agents, tested on seeds 0 to 9 for all three stationary environments.

Hyperparameter	Values
method	grid
seed	[0, ..., 6]
dynamic_rate	[200, 1000]
batch_size	[2, 5, 10, 20, 30, 50, 100, 200]
memory_size	[500, 1000, 1500, 2000]

Table 2: Non-stationary environment sweep hyperparameters

The relation between batch_size and observed mean_regret is shown in figures 5, 6 and 7. In stationary environments, memory_size was kept at 1000, while batch_size was reduced from 256 [11] to 50 for ContextualBandit-v2 and to 10 for NNBandit-v0. As shown in figure 5, no discernible difference could be observed between different memory_size values, though it is suggested that drastically smaller values would degrade performance. In non-stationary environments, which will be elaborated upon later in this section, dynamic_rate was the main deciding factor. Memory_size had slightly more influence, but not of any significance. At a dynamic_rate of 200, the lowest mean_regret was observed in ContextualBandit-v2

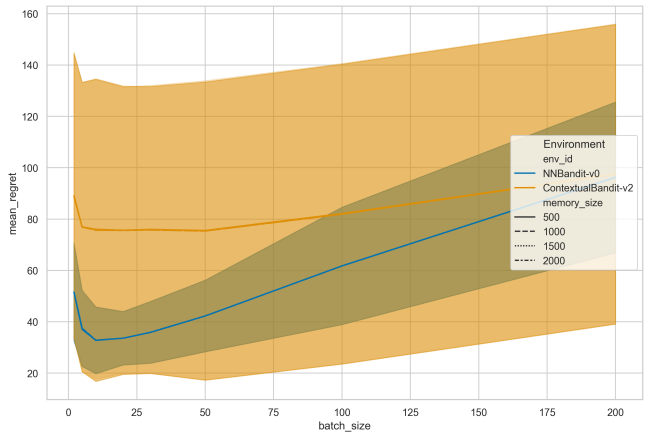


Figure 5: Hyperparameter sensitivity of batch_size in stationary environments

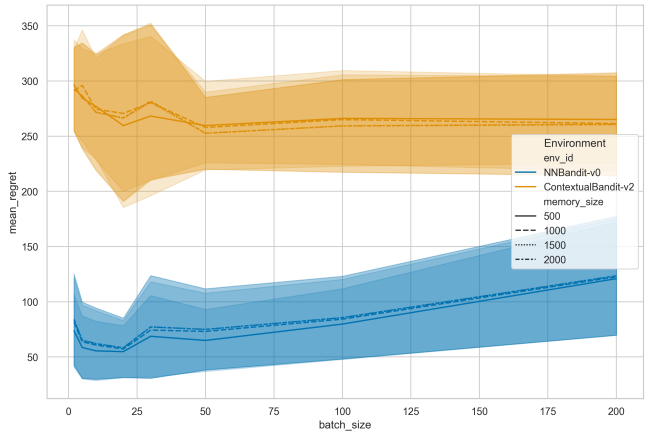


Figure 6: Hyperparameter sensitivity of batch_size in non-stationary environments with a dynamic_rate of 200

with a batch_size of 50, and in NNBandit-v0 with a batch_size of 20. At a dynamic_rate of 1000, values for both environments were comparable to their stationary counterparts.

Eventually the hyperparameters seen in table 3 were determined to be optimal across all three environments.

Parameter	**Value**
Episodes	2000
Optimizer	Adam
Hyperparameter	**Value**
hidden_layer_size	40
noisy_layer_distr_type	normal
noisy_layer_init_std	0.4
noisy_output_layer	true
batch_size	20
memory_size	1000

Table 3: General hyperparameters

To ensure a fair comparison between NoisyNet agents and those using ε-greedy exploration, it was necessary that the exploration rate of both was similar. First, 30 runs of the

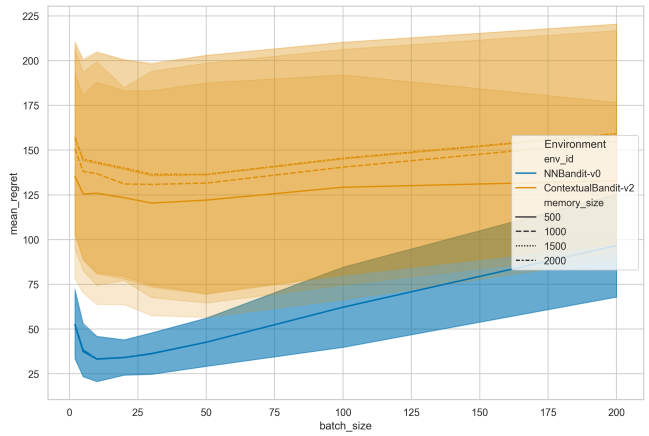


Figure 7: Hyperparameter sensitivity of batch_size in non-stationary environments with a dynamic_rate of 1000

NoisyNet agent were performed on every environment using different seeds. The resulting metrics showed that the mean exploration rate did not decay linearly. From this, approximately matching decay curves were derived for each environment, which are detailed in table 4 and 5. Using the same seeds as the NoisyNet-DQN runs, and these new epsilon decay functions, 30 runs were performed using regular DQNs. The observed score, regret and exploration rate, for both NoisyNet-DQNs and regular DQNs across all three environments, are shown in appendix B.1.

A second set of experiments was performed to compare performance on non-stationary environments, in which the reward structure changed during training. The dynamic_rate hyperparameter defined after what number of training steps this change occured. Two values were tested: 200 and 1000. The value of 1000 was chosen because it marks the midpoint of the 2000 total training steps, at which point most agents had converged to a stable policy. On the other hand, at 200 steps the agent's policy was only partially converged. At this point, the exploration rate was approximately half of its initial value.

Environment	**Decay function**
ContextualBandit-v2	$\varepsilon_{i+1} = \begin{cases} 0.38 & \text{for i} < 100 \\ 0.997\varepsilon_i & \text{for i} \geq 100 \end{cases}$
MNISTBandit-v0	$\varepsilon_{i+1} = \begin{cases} 0.4 & \text{for i} < 150 \\ \varepsilon_i - 0.002 & \text{for i} \geq 150 \end{cases}$
NNBandit-v0	$\varepsilon_{i+1} = \begin{cases} 0.35 & \text{for i} < 130 \\ 0.995\varepsilon_i & \text{for i} \geq 130 \end{cases}$

Table 4: ε decay functions for stationary environments

Environment	**Decay function**
ContextualBandit-v2	$\varepsilon_{i+1} = \begin{cases} 0.6 & \text{for i} = 0 \\ 0.9965\varepsilon_i & \text{for i} > 0 \end{cases}$
NNBandit-v0	$\varepsilon_{i+1} = \begin{cases} 0.6 & \text{for i} = 0 \\ 0.9965\varepsilon_i & \text{for i} > 0 \end{cases}$

Table 5: ε decay functions for non-stationary environments

5 Discussion

A consistent observation between measurements was that the MNISTBandit-v0 environment shows a higher mean regret compared to ContextualBandit-v2 and NNBandit-v0. This is most likely due to differences in how regret is calculated. In both ContextualBandit-v2 and NNBandit-v0 regret lies in the range $[0, 1]$, making the maximum regret 1. On the other hand, the MNISTBandit-v0 reward is binary, either 1 or -1, resulting in a maximum regret of 2. Furthermore, the denser rewards structure of ContextualBandit-v2 and NNBandit-v0 push regret values closer to zero. These factors could explain why the average regret in MNISTBandit-v0 was approximately three times higher.

Several hyperparameters were tested for their effect on the agent's performance. As concluded in the most general sweep, the noise sampling distribution (noisy_layer_distr_type) and whether the last layer of the agent was a NoisyLinear layer (noisy_output_layer), had no influence on the agent's performance across environments. The reward being offset by a random amount (noisy_reward), had too much influence. As such, this was treated as a separate problem type and excluded from further sweeps.

Hidden layer size (hidden_layer_size) showed minimal variation in performance across types of agents and environments.

Small values for noisy_layer_init_std led to high variation in mean regret, indicating unreliable behavior. The small perturbation it creates results in negligible exploration. This results in agents relying on luck to find an optimal policy during early training. This most drastically affected the ContextualBandit-v2 environment and minimally affected the MNISTBandit-v0 environment. All three environments found optimal performance between values 0.3 and 0.4.

Memory size had negligible effect in stationary environments and slight effects in non-stationary ones. Much smaller memory size than those tested is expected to degrade performance.

All environments had a drop in performance at batch sizes less than 5 due to insufficient training samples. ContextualBandit-v2 was largely unaffected by different batch sizes, with a slight preference to values between 30 and 50 in stationary and high dynamic-rate non-stationary environments. In low dynamic-rate settings, performance was consistent at batch size values of 50 and higher.

NNBandit-v0 performed best with batch size values between 10 and 20, with performance degrading at higher values.

In non-stationary environments with a dynamic rate of 1000, batch size and memory size hyperparameter sensitivity mirrored those of stationary settings.

It must be noted that the smoother curves seen in figures 5, 6 and 7 at higher values is the result of the lower resolution of tested values in those values.

As described in the introduction, NoisyNet was first demonstrated to, on average, improve performance on RL tasks [1]. As bandits can be considered simplified RL settings, being essentially one-state MDPs, similar improvement was hoped to be observed in this study.

The stationary ContextualBandit-v2 and NNBandit-v0 showed worse performance on average. Only in MNISTBandit-v0 did the NoisyNet agent outperform the baseline by finding an optimal policy more quickly. This suggests that NoisyNet works better in settings with high-dimensional inputs.

It should be noted that in non-stationary environments, a slight misalignment between the NoisyNet and non-NoisyNet exploration rates is present. In particular for ContextualBandit-v2, the small preference towards exploitation might explain the higher reward and lower regret.

In both stationary MNISTBandit-v0 and NNBandit-v0, regret plots show a faster convergence to optimal policies. However, in NNBandit-v0 (figure 11) the regular DQN overtakes the NoisyNet-DQN in score around 500 steps, indicating that the regular DQN finds better policies on average. This is also indicated by the slope of the regret curve.

6 Responsible Research

To ensure the robust results, all runs comparing results between NoisyNet-DQNs and regular DQNs, were performed using different random seeds. The number of runs for both hyperparameter sweeps and for comparison, were attempted to keep as high as possible, with the greatest limiting factor being computation time. To ensure transparency across all steps of the study, all experiments were logged using Weights & Biases. Sweeps were only deleted and re-executed in their entirety, either due to faulty parameters or incorrect sweep configuration. No individual runs were selectively removed from sweeps. All source code used in this project is publicly available at https://github.com/sonnyruff/testdqn, with references to outside software provided in the bibliography.

7 Conclusion and Future Work

This study evaluated the effectiveness of NoisyNet in contextual bandit settings, focusing on its sensitivity to hyperparameters, its performance across different task types, and the insights offered by visual evaluation metrics.

First, the hyperparameter sweeps showed that while most hyperparameters had minimal impact across environments, the NoisyNet specific initial standard deviation of the noisy layers (noisy_layer_init_std) had a significant effect. Small values reduced exploration and led to unreliable performance, especially in ContextualBandit-v2. Optimal performance was generally observed at values between 0.3 and 0.4 across all environments. The memory and batch size showed importance in slow-changing non-stationary, and stationary low-dimensional environment ContextualBandit-v2. Nevertheless, a set of task-independent hyperparameters that are generally applicable are seen in table 3.

Second, the performance analysis showed that NoisyNet-DQN does not universally outperform a standard DQN. While the stationary high-dimensional MNISTBandit-v0 environment showed improvement, the simpler ContextualBandit-v2 and NNBandit-v0 showed inferior results, in both stationary and non-stationary settings. This suggests that NoisyNet's strength lies in environments with a large state space.

Lastly, visual evaluation metrics such as regret and reward curves provided additional insights. The stationary MNISTBandit-v0 environment showed faster convergence to an optimal policy. While this improvement is also seen in the stationary NNBandit-v0 environment, the plot shows that the NoisyNet-DQN fails to find an optimal policy.

The original work of Fortunato [1] hoped to have their method be a step toward a universal exploration strategy. However, the high variance in policy performance observed in this study suggests that the method is potentially not robust enough to perform reliably in all environments. Its effectiveness shows in tasks with high-dimensional input spaces.

With the scope of this study having been limited by time constraints, we now look toward potential directions for future research. Several ideas were considered during the project but ultimately not explored.

- Initially, delayed reward settings were considered. However, incorporating this would require such a fundamental change to the MDP that it would result in a setting that could no longer be considered a contextual bandit setting, and would more closely resemble a traditional RL setting.
- Non-stationary environments were evaluated using hyperparameters tuned for stationary settings. Priority was given to implementing different environments. Since the specific non-stationary hyperparameter dynamic_rate was defined later in the study, the early hyperparameter sweeps did not account for it. Future work should define and include dynamic_rate from the start.
- Investigate visualization methods for high-dimensional input spaces. With dimension reduction and clustering techniques, plots similar to figures 1 and 2 could be created, offering insight into agent behavior.
- Smaller values for memory_size could be explored.
- Non-stationary environments with reward functions changing a very small amount each step, approaching a continuously changing reward.
- Comparing NoisyNet agents with non-NoisyNet agents that use different exploration decay functions, such as linear decay.

References

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

[2] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," Sept. 2022.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2 ed., 2018.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," May 1992.

[5] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-Bandit Approach to Personalized News Article Recommendation," in *Proceedings of the 19th international conference on World wide web*, pp. 661–670, Apr. 2010. arXiv:1003.0146 [cs].

[6] "Human-level control through deep reinforcement learning," *Nature*, vol. 518, Feb. 2015.

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

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

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

[10] H. Ishfaq, Q. Lan, P. Xu, A. R. Mahmood, D. Precup, A. Anandkumar, and K. Azizzadenesheli, "Provable and Practical: Efficient Exploration in Reinforcement Learning via Langevin Monte Carlo," Mar. 2024. arXiv:2305.18246 [cs].

[11] J. Park, "Rainbow is all you need." https://github.com/Curt-Park/rainbow-is-all-you-need, 2023.

[12] K. Arulkumaran, "Rainbow." https://github.com/Kaixhin/Rainbow/blob/master/model.py, 2020.

[13] Y. Dulat, "Rl-adventure: Noisy dqn." https://github.com/higgsfield/RL-Adventure/blob/master/5.noisy%20dqn.ipynb, 2018.

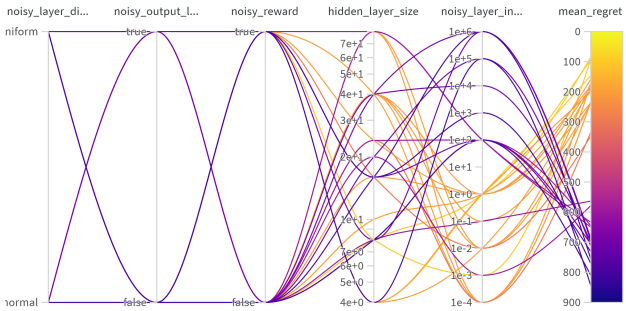
[14] A. Parker, "Buffalo gym." https://github.com/foreverska/buffalo-gym, 2025.

[15] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvári, S. Singh, B. Van Roy, R. Sutton, D. Silver, and H. van Hasselt, "Behaviour suite for reinforcement learning," in *International Conference on Learning Representations*, 2020.

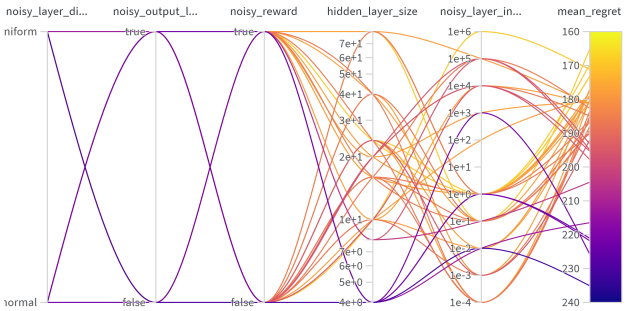
[16] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[17] Weights & Biases, "Parameter importance panel." https://docs.wandb.ai/guides/app/features/panels/parameter-importance/, 2024.

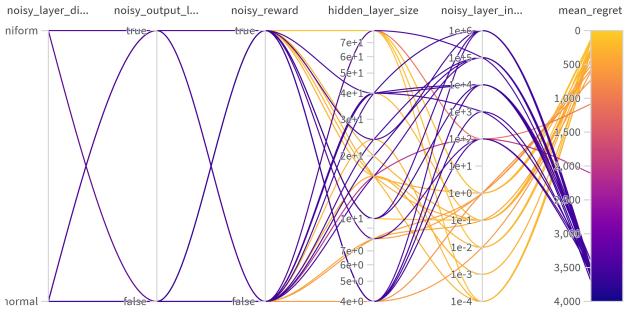
A Hyperparameter sensitivity



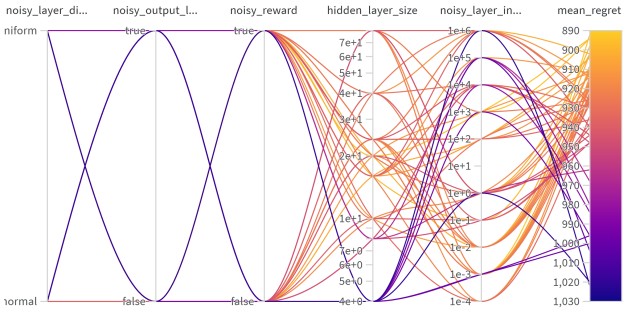
(a) ContextualBandit-v2 NoisyNet



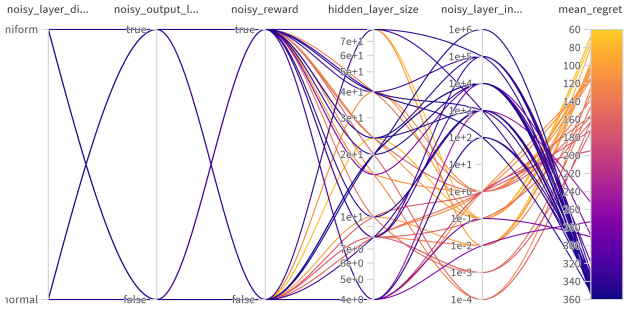
(b) ContextualBandit-v2 non-NoisyNet



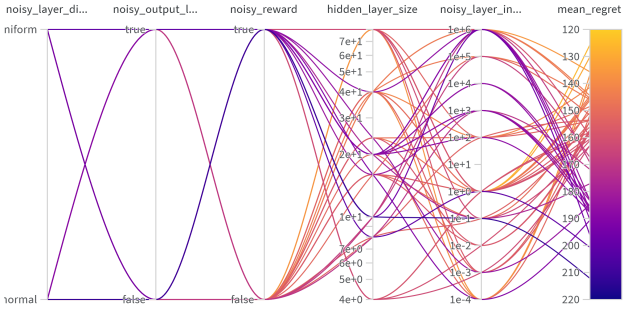
(c) MNISTBandit-v2 NoisyNet



(d) MNISTBandit-v2 non-NoisyNet



(e) NNBandit-v2 NoisyNet



(f) NNBandit-v2 non-NoisyNet

Figure 8: Parallel coordinate plots of Weights and Biases hyperparameter sweeps

Environment	**NoisyNet**	**Non-NoisyNet**
ContextualBandit-v2	`noisy_layer_init_std`	`hidden_layer_size`
MNISTBandit-v0	`noisy_layer_init_std`	`hidden_layer_size`
NNBandit-v0	`noisy_layer_init_std`	1st `noisy_reward`, 2nd `hidden_layer_size`

Table 6: Most sensitive hyperparameters for NoisyNet vs Non-NoisyNet across environments

B Results

B.1 Stationary Environments

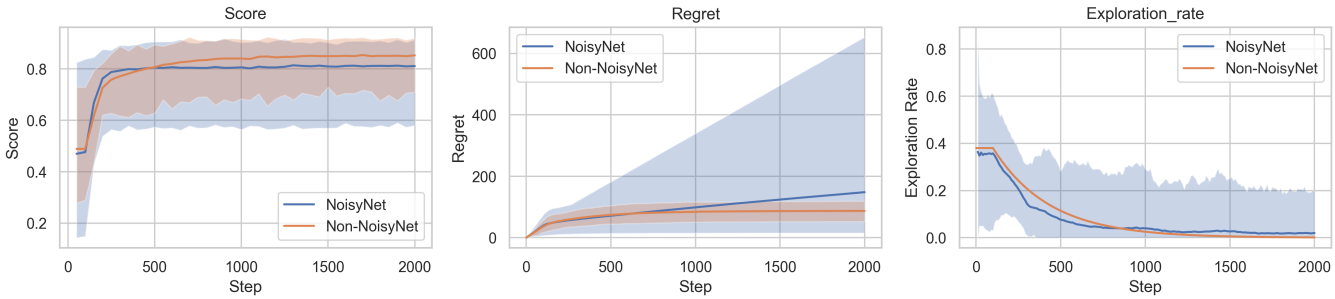


Figure 9: Stationary ContextualBandit-v2 metrics

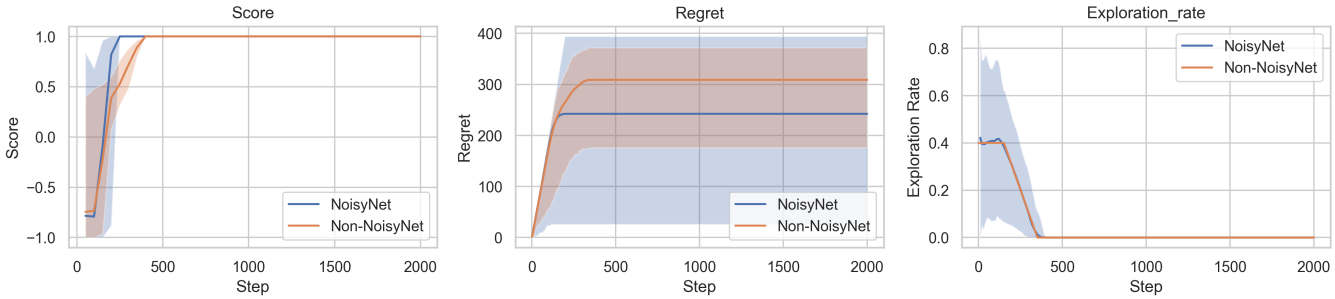


Figure 10: Stationary MNISTBandit-v0 metrics

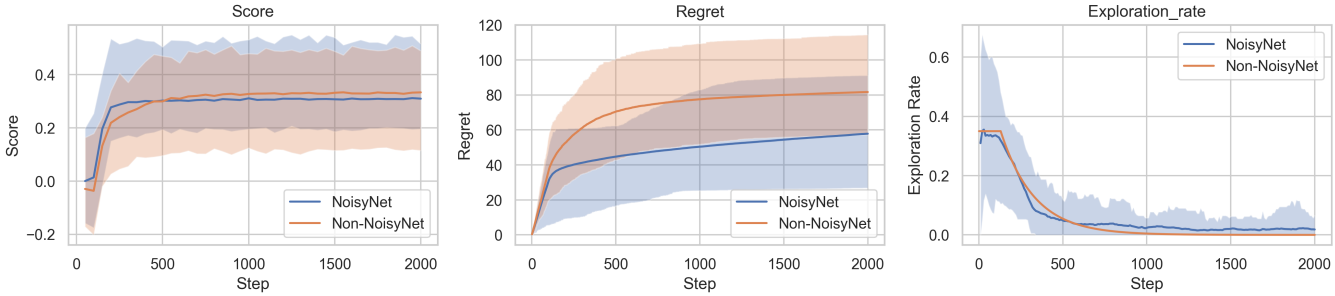


Figure 11: Stationary NNBandit-v0 metrics

B.2 Non-Stationary Environments

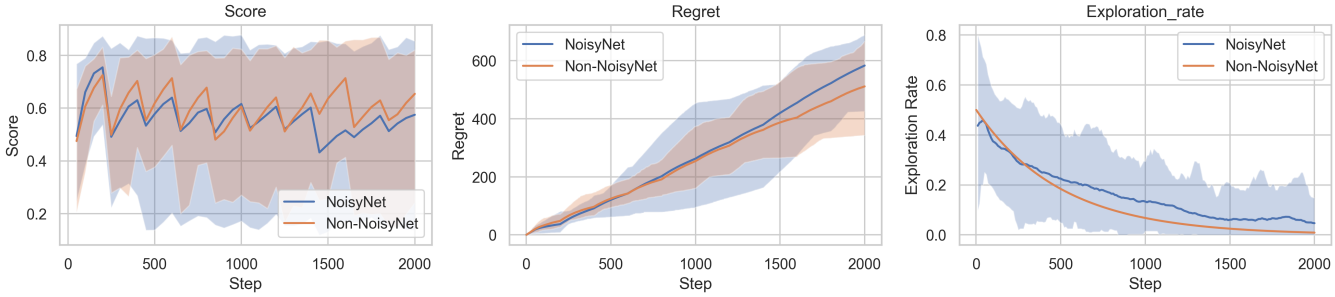


Figure 12: Non-Stationary ContextualBandit-v2 metrics; Dynamic rate set to 200 steps

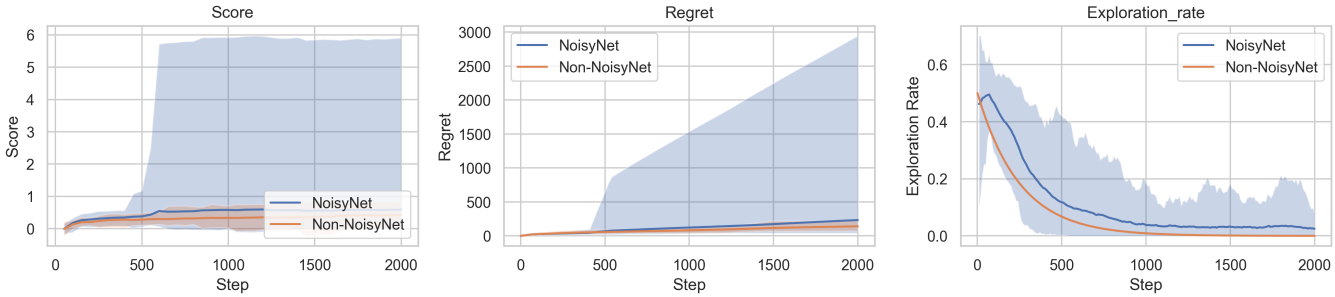


Figure 13: Stationary NNBandit-v0 metrics; Dynamic rate set to 200 steps

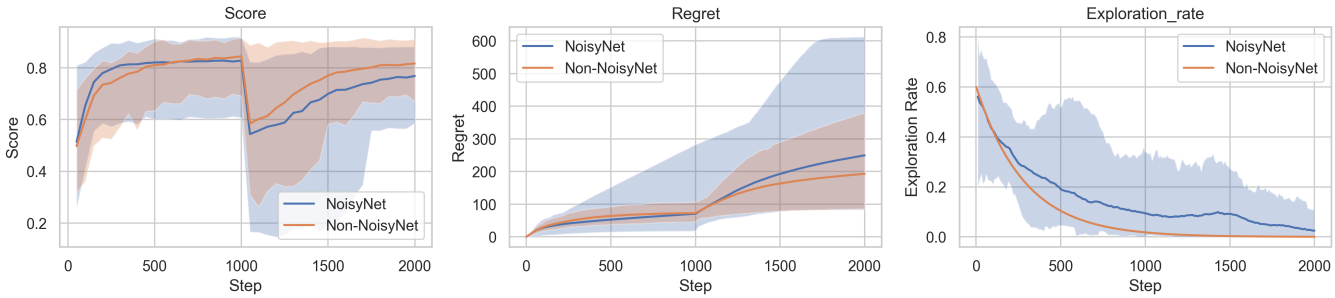


Figure 14: Stationary ContextualBandit-v2 metrics; Dynamic rate set to 1000 steps

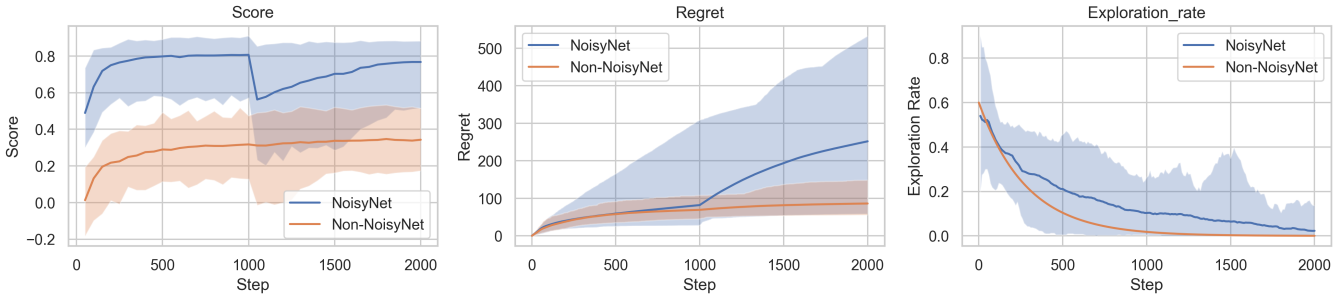


Figure 15: Stationary NNBandit-v0 metrics; Dynamic rate set to 1000 steps