# Biologically Inspired Method to Enhance Exploration: How Does Adding Noise to a Reinforcement Learning System Affect its Exploration?

**Alireza Vosoughi Rad** [*1] **Mahyar Karami** [*1] **Ali Shafiezadeh** [*2]

## Abstract

Exploration is a key challenge in reinforcement learning (RL), especially in complex environments where it is hard to define techniques for the agent so it can explore the whole environment to gain knowledge. In this project, we propose to investigate the idea of a biologically inspired method to enhance exploration, whose intuition is to add noise to the RL agent so that the agent makes non-greedy decisions and explores more states. We hypothesize that adding noise to the output Q-values of the neural network can improve the exploration and performance of deep RL (D-RL) agents. We plan to test our hypothesis on DQN in a grid world maze environment and compare it with existing exploration methods. We will show that the proposed method outperforms the conventional solutions in the coverage sense. However, our method could not outperform epsilon-greedy and NoisyNet algorithms in the sense of convergence. The codes for the project can be found in the GitHub repository https://github.com/Alishafzd/Noisy-DQN.

## 1. Introduction

D-RL is a powerful framework for learning policies from data and experience. However, agents often face the dilemma of exploration versus exploitation: how to balance between exploiting the current knowledge and exploring new possibilities. Exploration is crucial for discovering novel states and actions, obtaining diverse and informative feedback, and avoiding local optimal or suboptimal policies. However, exploration is also challenging, especially in high-dimensional and complex environments, where rewards are sparse, delayed, or deceptive, and where the state and action spaces are large or continuous (Hao et al., 2023).

Existing exploration methods rely on extrinsic or intrinsic motivation, such as reward shaping (Yuan et al., 2023), entropy maximization (Seo et al., 2020), curiosity-driven learning (Zheng et al., 2021), or information gain (Fortunato et al., 2018). However, these methods often require additional computation or estimation, which can be costly or inaccurate. Moreover, these methods may not capture the full spectrum of exploration behaviors that natural systems exhibit. In a recent study, Taiga et al. (2021) evaluated four different bonus-based exploration methods for reinforcement learning agents: count-based bonus, prediction error bonus, random network distillation bonus, and information gain bonus. They used Rainbow (Hessel et al., 2018), a state-of-the-art deep Q-network architecture, as the base model for all the methods and tested them on a variety of Atari 2600 games (Bellemare et al., 2013). They found that exploration bonuses improved the performance of Montezuma's Revenge, a notoriously hard exploration game. However, they did not provide significant gains over epsilon-greedy exploration, the default scheme in Rainbow, on other hard Atari 2600 games.

Fortunato et al. (2017) have proposed NoisyNet, a deep reinforcement learning agent that adds parametric noise to its weights, allowing it to adapt its exploration strategy to the task and the environment. They use three types of deep reinforcement learning agents, A3C (Mnih et al., 2016), DQN (Mnih et al., 2015), and Dueling architectures (Wang et al., 2016), two types of environments, Atari 2600 games and Mujoco continuous control tasks (Todorov et al., 2012), and two types of noise functions, factorized Gaussian noise and independent Gaussian noise. The study injects noise at two steps: before and after the activation function, and compares the results for each option. They found that injecting noise before the activation function works better for A3C and DQN, while injecting noise after the activation function works better for dueling. The paper also shows that NoisyNet outperforms the conventional exploration heuristics on most of the tasks, achieving state-of-the-art or super-human performance on some of them. However, they do not investigate the effect of adding noise to the loss function or the environment reward.

In this study, we aim to explore a biologically-inspired approach to enhance exploration in reinforcement learning. It is well-documented that neuronal activity in the brain is not entirely quiescent but exhibits random fluctuations in firing rates (Destexhe et al., 2007). These fluctuations, often perceived as noise, are typically filtered out during the analysis of dopamine activations. However, we posit the question: could this inherent noise play a significant role in exploration? We propose to train a D-RL agent using a stan-

dard loss function, such as that used in Deep Q-Networks (DQN), with the addition of Gaussian noise at each time step to the output Q-values. The underlying hypothesis is that the introduction of noise to the Q-values will induce errors in the gradient. Consequently, the model, trained with this noisy gradient, may make decisions that deviate from the greedy policy, thereby encouraging the exploration of a broader range of states. The injected noise could serve a dual purpose by acting as a regularizer, thereby mitigating overfitting and enhancing the model's generalization capabilities (Bishop, 1995). This study aims to provide empirical evidence verifying these hypotheses.

We hypothesize that our method can improve the exploration and performance of D-RL agents on various tasks and environments. We plan to test our hypothesis by conducting experiments using Gaussian noise added to output Q-values of the neural network with different approaches in DQN in a grid world maze which is a simple environment to visualize and analyze results.

In the following sections of this paper, we will delve deeper into our proposal. In Section 2, we will review some basic concepts in RL, D-RL, and exploration methods. In Section 3, we define the main question for the project. Section 5 is devoted to describing the environment we want to use for our empirical experiments to test our hypothesis, defining the evaluation metrics used in this project, and determining the structures and parameters of simulation. Numerical results and their analysis would be in Section 6. Finally, we will conclude the whole project in Section 7.

## 2. Background

### 2.1. Reinforcement Learning

RL is a powerful computational approach for learning to optimize behavior from experience. In the RL framework, an agent acquires experience by interacting with an environment through a sequence of observations, actions, and rewards. At each discrete time step $t$, the agent observes a state $s_t$ from the state space $\mathcal{S}$, which represents the current situation or condition that the agent finds itself in. The agent then chooses an action $a_t$ from the action space $\mathcal{A}$ according to a policy $\pi(a_t|s_t)$, which is a probabilistic mapping from states to actions. Upon taking this action, the agent receives a scalar reward $r_t$ from the reward function $\mathcal{R}(s_t, a_t)$, which quantifies the immediate benefit of taking action $a_t$ in state $s_t$. The agent also transitions to a new state $s_{t+1}$ according to the transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$, which describes the dynamics of the environment. The ultimate goal of the agent is to learn a policy that maximizes the expected return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, where $\gamma \in [0, 1]$ is a discount factor that trades off immediate and future rewards. This cumulative discounted return serves as a measure of long-term performance and success under policy $\pi$.

### 2.2. Deep Reinforcement Learning

D-RL stands as a remarkable subset within the field of RL, leveraging the power of deep neural networks to represent critical components of the RL process, including value functions, policies, and even models. This amalgamation of RL and deep learning has revolutionized the landscape of autonomous decision-making, enabling agents to tackle highly complex and high-dimensional environments that were previously unattainable using traditional RL methods.

D-RL harnesses deep neural networks to approximate value functions, such as $Q(s, a)$, which estimate the expected return associated with taking specific actions from states. Furthermore, these networks can also approximate policies, denoted as $\pi(a|s)$, offering a streamlined mapping from states to actions. By employing neural networks, D-RL agents can handle a vast array of states and actions effectively. One of the most defining features of D-RL is its aptitude for dealing with intricate, high-dimensional environments. This includes tasks involving image or video inputs, natural language understanding, and other sensory-rich data. The ability to process and learn from such data empowers agents to navigate real-world scenarios, such as autonomous vehicles interpreting camera feeds or language models responding to diverse user queries.

Some examples of D-RL algorithms are:

- Deep Q-Network (DQN) (Mnih et al., 2015), which uses a deep neural network to approximate the Q-function and applies experience replay and target network to stabilize the learning.
- Double Deep Q-Network (DDQN) (Van Hasselt et al., 2016), an extension of DQN that mitigates overestimation biases by using two separate networks for action selection and action evaluation. DDQN addresses the issue of overestimating Q-values, leading to more stable and accurate learning.
- Advantage Actor-Critic (A2C) (Mnih et al., 2016), which uses two networks: an actor network that outputs a stochastic policy and a critic network that outputs a state-value function. The actor network is updated by policy gradient with an advantage function computed by the critic network.
- Soft Actor-Critic (SAC) (Haarnoja et al., 2018), which extends the actor-critic framework with an entropy regularization term that encourages exploration and robustness. The actor network outputs a stochastic policy that maximizes both the expected return and the entropy. The critic network outputs two Q-functions that are updated by minimizing the Bellman residual and enforcing consistency.

Despite its capabilities, D-RL faces several challenges that researchers continuously strive to overcome. One of significant challenges in D-RL is the exploration issue, which

is the main topic of the project. Balancing exploration and exploitation remains a key challenge in D-RL. Agents need to explore the environment to discover optimal policies, but excessive exploration can hinder learning progress. So, exploration-exploitation also stands for D-RL agents and maybe it is harder to make a balance here.

### 2.3. Exploration Methods

Exploration is a key challenge in RL, especially in complex and sparse environments. Exploration methods can be categorized into two types: extrinsic and intrinsic.

Extrinsic methods use external rewards or information to guide exploration. For example, Tang et al. (2017) presents a simple approach for exploration, which extends classic counting-based methods to high-dimensional, continuous state spaces1. States are mapped to hash codes, which allows to count their occurrences with a hash table. These counts are then used to compute a reward bonus according to the classic count-based exploration theory. This exploration strategy achieves near state-of-the-art performance on both continuous control tasks and Atari 2600 games.

Intrinsic methods use internal rewards or information to guide exploration. One example for these methods is curiosity-driven learning (Pathak et al., 2017), which uses a prediction error of a learned model of the environment as an intrinsic reward that motivates the agent to explore novel and uncertain states. We can also categorize information gain (Houthooft et al., 2016) in this type of methods. Information gain methods measure the reduction in uncertainty or surprise about the environment or the agent's own behavior as an intrinsic reward that drives the agent to learn more about its surroundings. One more example of these type of methods is random network distillation (Burda et al., 2018), which uses the error between a randomly initialized neural network and another neural network trained to mimic it as an intrinsic reward that reflects the novelty of the states.

The use of noise for enhancing exploration in D-RL can be considered as an intrinsic method. This is because the noise is directly incorporated into the agent's learning process, specifically into the agent's parameters or actions, rather than being an external factor that influences the reward or environment. The noise encourages the agent to explore different parts of the state-action space, by not choosing the greedy action, leading to more diverse and potentially more effective policies. This is in contrast to extrinsic methods, which typically involve modifying the reward structure or environment to encourage exploration.

In recent years, several works have explored the use of noise for enhancing exploration in D-RL. A notable example is the NoisyNet proposed by Fortunato et al. (2017), which adds parametric noise to the weights of deep reinforcement learning agents, thereby adapting their exploration strategy to the task and environment. They found that the location of noise injection impacts the performance of different types of agents. Hollenstein et al. (2023) provide a comprehensive analysis of action noise in off-policy D-RL for continuous control. They investigate the impact of different types of action noise, such as Gaussian and Ornstein-Uhlenbeck noise, on the learned policy. Plappert et al. (2018) discuss how adding temporally correlated noise can enhance the exploratory nature of D-RL algorithms. It demonstrates that parameter noise leads to better exploration by generating a policy that exhibits a larger variety of behaviours.

## 3. Research Question

Our research question is: **Does adding the noise to the Q-values enhance the exploration of D-RL agents on various tasks and environments?**

We aim to answer this question by conducting experiments that Gaussian noise is added to the output Q-values of the neural network in different ways on a DQN agent and a grid world maze environment. We will compare our results with existing exploration methods (e.g., epsilon-greedy).

We expect to find that adding noise can make required randomness in the absence of epsilon so the agent can explore the environment. We also expect to gain insights into how adding noise affects the learning process and outcome of the D-RL agents.

## 4. NoisyQ for Reinforcement Learning

In the NoisyQ, we introduce a novel approach to enhance the performance of Deep Double Q-Networks (DDQN) by incorporating controlled Gaussian noise into the network's output. Inspired by the principles of NoisyNets (Fortunato et al., 2017), we manipulate the Q-values with carefully designed noise injections. The objective is to introduce stochasticity into the decision-making process of the agent, promoting exploration and robust learning.

### 4.1. NoisyQ Network Architecture

Our NoisyQ network modifies the standard DDQN by introducing controlled Gaussian noise to the Q-values for the sake of randomness required to explore the environment. Let $\hat{y}$, which is calculated by the target network, represent the output of the network, i.e., the Q-values. We redefine the NoisyQ output as follows:

$$\hat{y}_{\text{NoisyQ}} = \hat{y} + \sigma \cdot \vec{\varepsilon}$$

Where:

- $\hat{y}_{\text{NoisyQ}}$: The NoisyQ modified Q-values.
- $\hat{y}$: The original output of the DDQN.
- $\sigma$: A constant variance parameter.
- $\vec{\varepsilon}$: A random variable vector drawn from a Gaussian distribution with zero mean and unit variance.

Then, the loss is calculated with the new output of the network or $\hat{y}_{\text{NoisyQ}}$, and it is used for learning the network.

### 4.2. NoisyQ Variants

#### 4.2.1. NOISYQ ALL-Q

In this variant, we add Gaussian noise to all Q-values produced by the DDQN. This introduces controlled randomness to the entire set of action values, influencing the exploration-exploitation trade-off during training.

$$\hat{y}_{\text{NoisyQ All-Q}} = \hat{y} + \sigma \cdot \vec{\varepsilon}$$

In this scenario, the Gaussian noise may or may not change the maximum Q-value selected for calculating the target and this definitely change. Hence there would be a possibility of absence of noise itself in the loss function.

#### 4.2.2. NOISYQ MAX-Q

This variant focuses on perturbing only the maximum Q-value for a given state. The intent is to add variability to the network's most confident prediction, potentially encouraging the agent to explore alternative actions.

$$\hat{y}_{\text{NoisyQ Max-Q}} = \hat{y} + \vec{E}$$

Where $\vec{E}$ is a vector that All of its elements are zero except the one with the index of $\max(\hat{y})$. The non-zero element is $\sigma \cdot \vec{\varepsilon}$.

#### 4.2.3. NOISYQ AVG-Q

Here, we take the average of all Q-values produced by the DDQN and add Gaussian noise to this averaged value. This approach aims to introduce a systemic perturbation across all actions, fostering exploration in a more balanced manner.

$$\hat{y}_{\text{NoisyQ}} = \text{mean}(\hat{y}) + \sigma \cdot \vec{\varepsilon}$$

Actually by adding noise in this way, we are forcing the network tune it's weight so the output would be equal in an average manner but with a variance difference which makes the agent to decide randomly and explore the environment. Meanwhile, it is noteworthy that we stop this method after some steps we are sure that agent has explored the hole environment for the sake of convergence. Because in this method we are changing the average too.

### 4.3. NoisyQ Loss

Let $\theta$ represent the set of learnable parameters in our NoisyQ network. The modified loss function $\mathcal{L}_{\text{NoisyQ}}(\theta)$ is computed as the Mean Squared Error (MSE) between the predicted Q-values ($\hat{y}_{\text{NoisyQ}}$) and the target Q-values:

$$\mathcal{L}_{\text{NoisyQ}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^i_{\text{NoisyQ}} - y^i)^2$$

where:

- $N$: The number of samples in the mini-batch.
- $\hat{y}^i_{\text{NoisyQ}}$: The $i^{th}$ element of the NoisyQ modified Q-values.
- $y^i$: The target Q-value for the $i^{th}$ sample.

The NoisyQ modified Q-values depend on the specific variant used. The Gaussian noise term $\vec{\varepsilon^i}$ is a random variable drawn from a distribution with zero mean and unit variance.

**Gradients Calculation**. To optimize the NoisyQ network, we employ the stochastic gradient descent (SGD) algorithm. The gradients with respect to the learnable parameters $\theta$ are calculated using a Monte Carlo approximation. Specifically, the gradients are approximated as:

$$\nabla\mathcal{L}_{\text{NoisyQ}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla(\hat{y}^i_{\text{NoisyQ}} - y^i)^2$$

where:

- $\nabla$: Denotes the gradient operator.
- $N$: The number of samples in the mini-batch.

This Monte Carlo approximation allows us to efficiently estimate the gradients of the modified loss function, facilitating the training of the NoisyQ network. The optimization process seeks to adapt the learnable parameters $\theta$ to improve the agent's decision-making capabilities under the influence of controlled stochasticity introduced by the NoisyQ variants.

### 4.4. NoisyQ-DDQN

In this section, we elucidate the modifications applied to the Deep Double Q-Network (DDQN) to seamlessly integrate our innovative NoisyQ methods: NoisyQ All-Q, NoisyQ Max-Q, and NoisyQ Avg-Q.

**Policy Adjustment**. Departing from the traditional DDQN approach, we discard the $\epsilon$-greedy exploration strategy. Instead, the policy is refined to optimally exploit the (randomized) action-value function with the introduced controlled Gaussian noise.

**Fully Connected Layers**. The last fully connected layer (output layer) of the online network is now parameterized as a noisy layer. After each replay step, based on one of the methods described in section 4.2, we add noise to predicted Q-values of the online network.

**Loss Function for NoisyQ-DDQN.** The loss function for our NoisyQ adaptation of DDQN, termed NoisyQ-DDQN, is derived to suit the modifications introduced by NoisyQ. The action-value function $Q(x, a, \hat{y}, \theta)$, where $\hat{y}$ represents the NoisyQ modified Q-values, is treated as a random variable. The NoisyQ-DDQN loss is defined as:

$$\mathcal{L}_{\text{NoisyQ-DDQN}}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{b \in A} Q(x, b, \theta^-) - Q_{\text{NoisyQ}}(x, b, \theta)\right)^2\right]$$

Where:

- $x$: Represents the state that is given to the NoisyQ-DDQN.
- $\theta$: The set of learnable parameters in our NoisyQ-DDQN.
- $\theta^-$: The set of target network parameters.
- The outer expectation is with respect to the distribution of the NoisyQ parameters.

**Gradients Calculation.** To optimize the learnable parameters $\theta$, we utilize a Monte Carlo approximation for the gradients of the NoisyQ-DDQN loss:

$$\nabla\mathcal{L}_{\text{NoisyQ-DDQN}}(\theta) \approx 2 \cdot \mathbb{E}\left[\left(r + \gamma \max_{b \in A} Q(x, b, \theta^-) - Q_{\text{NoisyQ}}(x, b, \theta)\right) \cdot \nabla Q_{\text{NoisyQ}}(x, b, \theta)\right)\right]$$

The NoisyQ-DDQN algorithm is provided in Appendix 9.1.

### 4.5. Recap

The introduction of controlled noise in the NoisyQ variants aims to address the challenge of the exploration-exploitation trade-off in reinforcement learning. By perturbing the Q-values with controlled randomness, we provide the agent with the opportunity to explore alternative actions, potentially leading to more effective policy learning. It is noteworthy to say that due to the effect of learning rate on how much noise changes the weights, we should initialize it with a big number, then decrease it slowly so the network can converge.

Our experimental setup involves applying NoisyQ variants to DDQN and evaluating their impact on learning efficiency, stability, and overall performance. We hypothesize that the introduction of controlled noise will enhance the exploration capabilities of the agent, allowing for more robust learning in complex environments. The effectiveness of each NoisyQ variant will be assessed through comprehensive empirical evaluations.

## 5. Experimental Design and Evaluation Approach

This section is devoted to describe of how we test the hypothesis and evaluate the results. First we will describe the environment and methodology to experiment our hypothesis. Then, we will introduce approaches and metrics for evaluating the results.

### 5.1. Environment

As previously stated, the experimental evaluations are conducted within a grid-world maze. The choice of this environment is motivated by its inherent simplicity, which facilitates the tracking of the agent and the measurement of performance metrics pertinent to exploration. The specific environment employed in our study is a $16 \times 16$ grid-world, the layout of which is depicted in Figure 1. As depicted in the figure, the starting coordinates is $(1, 1)$ and the agent must reach the cell with coordinates $(16, 16)$ to terminate the episode.

The state representation for this environment is encoded as a three-layer tensor. The first layer of this tensor represents free cells, with corresponding elements in the matrix set to one. The second layer corresponds to obstacle cells, also indicated by ones in the respective matrix elements. The third layer signifies the agent's cell, marked by one in the corresponding matrix element. These three layers are stacked to form the input tensor, which is subsequently fed into the neural network.

It is important to highlight that the reward assigned for each step taken by the agent is set to zero. This approach is adopted to avoid the effect of optimistic initialization on exploration because we want just to see the effect of our methods in exploration. Upon reaching the goal cell, the agent is rewarded with a value of one. The reward function can be formulated as the following:

$$\text{reward}(s, a, s') = \begin{cases} 0 & \text{if } s' \text{ is not the goal state} \\ 1 & \text{if } s' \text{ is the goal state} \end{cases}$$

Where:

- $s$: Represents the current state of the agent.
- $a$: Denotes the action taken by the agent.
- $s'$: Represents the resulting state after taking action $a$.

### 5.2. Network Structure

The architecture of the neural network utilized in our proposed algorithm, NoisyQ-DDQN, is outlined in Table 1. The initial layer is a convolutional layer, a direct consequence of the state representation discussed earlier. Following this, the output undergoes MaxPooling and flattening operations before entering a fully connected network, the parameters of which are detailed in the table.

**NoisyQ** and **NoisyNet** share the same neural network structure. The primary distinction lies in the application of noise. In NoisyQ, Gaussian noise with mean zero and a varying variance is introduced to the output Q-values of the neural network. The noise reduction rate (NRR) governs the reduction of this noise, contributing to convergence. On the other hand, **epsilon-greedy** follows a similar structure

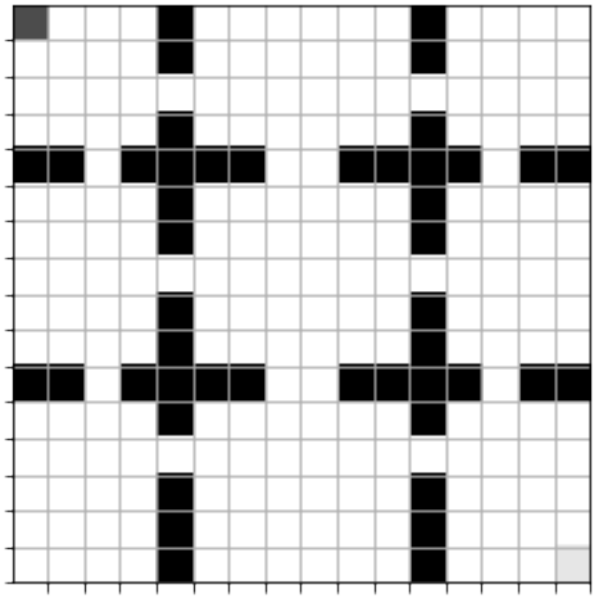

*Figure 1.* The selected grid world maze utilized for agent training.

| Layer Type | Output Shape | # of Parameters |
|---|---|---|
| Input | (None, 16, 16, 3) | 0 |
| Conv2D | (None, 14, 14, 6) | 54 |
| MaxPooling2D | (None, 7, 7, 6) | 0 |
| Flatten | (None, 294) | 0 |
| Dense | (None, 32) | 9440 |
| Dense | (None, 4) | 132 |

*Table 1.* Summary of the NoisyQ-DDQN neural network structure

with epsilon-greedy exploration, where epsilon gradually decays from its maximum to minimum value, influencing the agent's exploration-exploitation strategy.

The parameters employed in the simulations, including the discount factor ($\gamma$), epsilon decay, and Gaussian noise parameters, are detailed in Table 2. The choice of $\gamma$ less than one is deliberate to expedite the agent's discovery of the optimal path, considering the impact of the reward function detailed in Section 5. The NRR governs the reduction in noise variance to facilitate convergence.

### 5.3. Hyperparameter Search Space

The parameters utilized in our simulations are detailed in Table 2. The discount factor ($\gamma$) is set to 0.9, chosen to expedite the agent's discovery of the optimal path by emphasizing the importance of immediate rewards. The epsilon ($\epsilon$) parameter, governing the agent's exploration-exploitation strategy, decays from 1 to 0.0001 over the specified range. The epsilon decay, determining the rate of this decay, ranges from 1000 to 10000 steps.

Gaussian noise, introduced to the output Q-values of the neural network, has a mean of zero and a varying variance ($\sigma$), with the range specified in the table. To facilitate convergence, the noise is reduced based on the noise reducing rate (NRR) through the relation $\sigma \leftarrow \frac{\sigma}{\text{ceil}(step/NRR)}$.

The learning rate ($\alpha$), influencing the extent to which noise alters the weights, is gradually decreased from its maximum to its minimum value, with the range provided in the table.

| Parameter | Value/Range |
|---|---|
| $\gamma$ | 0.9 |
| $\epsilon$ | 1 - 0.0001 |
| Epsilon Decay | 1000 - 10000 |
| $\sigma$ | 0.1 - 1 |
| NRR | 500 - 5000 |
| $\alpha$ | 0.05 - 0.0005 |

*Table 2.* Shared hyperparameter search space for the NoisyQ-DDQN, NoisyNet-DDQN, and epsilon-greedy-DDQN. Note: NoisyNet-DDQN and epsilon-greedy-DDQN do not utilize epsilon. NoisyQ-DDQN does not use epsilon and follows a similar structure to NoisyNet-DDQN with additional noise applied.

### 5.4. Evaluation Approaches

In assessing the performance and exploration capabilities of our agent, we employ a set of comprehensive metrics focusing on episode termination, coverage, and exploration entropy.

**Episode per Step.** A performance metric is the count of terminated episodes ($k$), representing the number of steps taken by the agent to reach the goal cell. Optimal path convergence implies a constant ratio of terminated episodes relative to the step count. Therefore, plotting the number of terminated episodes against the step count should exhibit a linear relationship. The faster attainment of this linear relationship signifies expedited convergence.

**Coverage.** To quantify exploration, we introduce the coverage matrix, mirroring the environment's dimensions. As the agent explores each cell, the corresponding element in the coverage matrix is incremented. Visualizing this matrix as a heatmap provides insights into the agent's exploration pattern. Dynamic changes in the heatmap denote effective exploration, while static regions signal areas requiring further investigation.

**Entropy.** To encapsulate the information within the coverage matrix, we compute its entropy. The entropy of a random variable with a probability mass function $p(x)$ is given by $-\sum_{x\in\mathcal{X}} p(x)\log(p(x))$. Maximum entropy occurs when the random variable follows a uniform distribution, analogous to optimal coverage where all regions are explored uniformly. Computing the entropy of the normalized coverage matrix's elements allows us to quantify the quality of exploration. Higher entropy values indicate

superior exploration quality.

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log(p_i)$$

where $p_i$ is the normalized probability of the agent being in cell $i$ and $n$ is the total number of cells in the environment.

## 6. Results

In this section, we present a qualitative and quantitative comparison of our method (NoisyQ) with two baselines: epsilon greedy and noisy net. The comparisons are based on our metrics including the coverage, entropy, and episodes per steps. We train each agent for 10 times with different random seeds and report the average results. For NoisyQ, we tune the noise parameters sigma and NRR over three values each: $\sigma \in [0.1, 0.5, 1]$ and $NRR \in [500, 1000, 5000]$. We record the coverage matrix at three checkpoints during the training: steps 5000, 10000, and 15000. We compute the entropy for each coverage matrix and present the results in the appendix in different tables for each method.

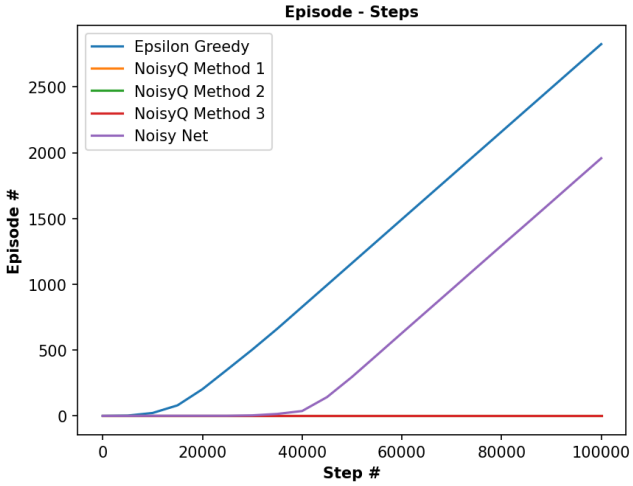


*Figure 2.* Episode per step for different methods. The constant tangent of the line indicates convergence.

Figure 3 shows the heat map of the coverage matrix for each method, including three NoisyQ models, epsilon greedy, and noisy net. The entropy values are displayed below each heat map. We select the best model for each method based on the highest entropy value across the checkpoints. For more information, the entropy table for each method is presented in the appendix, and the selected model is shown in boldface in the table. This figure demonstrates that NoisyQ models outperform the baselines in terms of exploration, especially at the first two checkpoints. However, Figure 2 reveals that NoisyQ models have problem converging to the optimal point at the end, indicating that they are not efficient in learning.

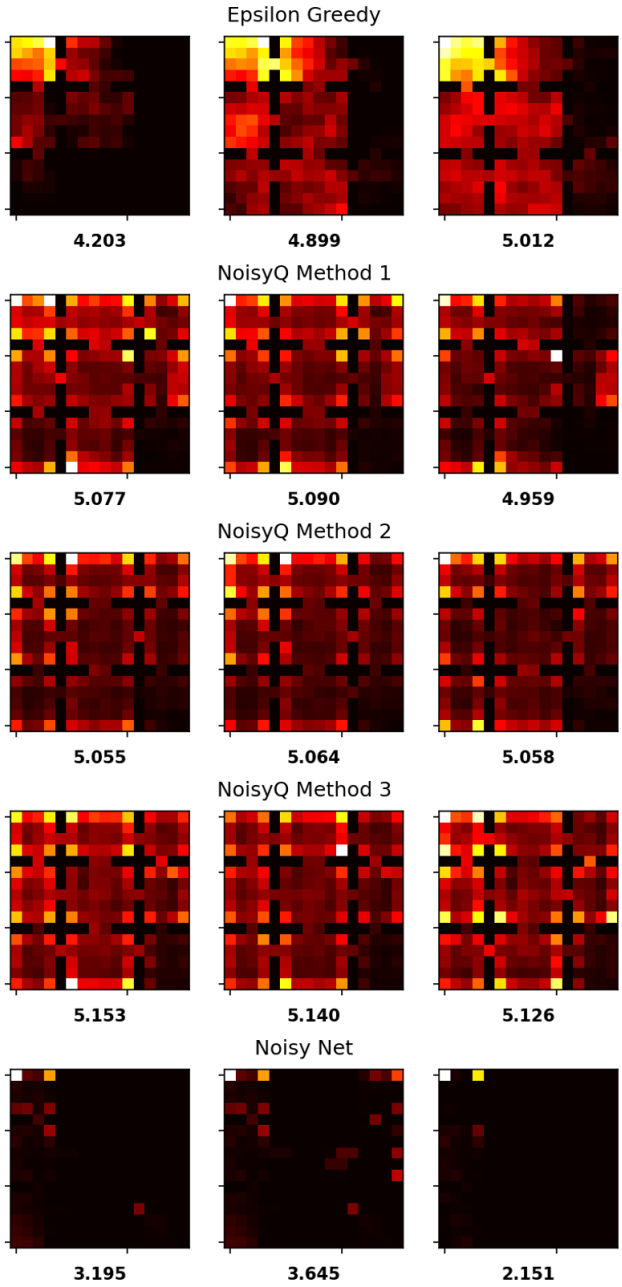


*Figure 3.* Heat maps for the best models of each method.

One possible explanation for this issue is that the agent gets stuck in a local minimum. To see the effect of adding noise to the loss, we need a large learning rate (we start from 0.5). But to converge, we also need to reduce the learning rate gradually. The noise and randomness enhance the exploration in the early stages, but they also make the weights fall into a local minimum that the agent cannot

escape with small learning rates (of the order of $10^{-4}$). The advantage of epsilon greedy is that the randomness does not directly affect the weights, as it only influences the action selection.

Another parameter that might affect the convergence of the DDQN model is the noise architecture. The noise architecture determines the distribution and the complexity of the noise that is added to the loss function. If the noise is too simple or too diverse, it can lead to either insufficient or excessive exploration. A simple noise function, such as a Gaussian function, might produce noise that is too random and uncorrelated, resulting in the agent exploring too many irrelevant or suboptimal states. Therefore, using a noise network architecture that can be trained along with the main network might be beneficial, as it can generate noise that is more adaptive and optimal for exploration. This might explain why the noisy net model converges better than the NoisyQ model. However, this idea is beyond the scope of a course project and can be pursued in our future research in this area.

## 7. Conclusion

In this paper, we explored the idea of adding noise to the Q-values of a D-RL agent to enhance its exploration in a grid-world maze environment. We proposed three different methods of adding noise to the loss function of a DDQN algorithm and compared them with two baselines: epsilon greedy and noisy net. We evaluated the performance of each method using three metrics: coverage matrix, entropy, and episodes per step. Our results showed that adding noise to the Q-values improved the coverage and entropy of the agent, but also reduced its efficiency and convergence. We analyzed the possible reasons for this trade-off and suggested some directions for future research. We hope that our work can inspire more research on noisy exploration methods for D-RL.

## 8. Contributions

**Alireza Vosoughi Rad:** Contributed to the project through a part of the literature review, the collaborative proposal of methods with other team members, critical analysis of proposed methods, project management, active involvement in implementing conceptual ideas, execution of NoisyNet implementation, the whole background and methodology sections of the report, and a collaboration on experiment section of the report.

**Mahyar Karami:** Contributed to the project through a part of the literature review, the collaborative proposal of methods with other team members, critical analysis of proposed methods, active involvement in implementing conceptual ideas, implementing the DDQN and NoisyQ-DDQN, writing the abstract, introduction section and collaborating on experimental design section.

**Ali Shafiezadeh:** Contributed to the project through a part of the literature review, the collaborative proposal of methods with other team members, critical analysis of proposed methods, active involvement in implementing conceptual ideas, implementing the DDQN and NoisyQ-DDQN, writing results and conclusion parts and also collaborating in the introduction and background parts of the report, and the key contributor to parameter search experiment.

## References


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

Bishop, C. M. Training with noise is equivalent to tikhonov regularization. Neural Computation, 7(1):108–116, 1995.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. In International Conference on Learning Representations, 2018.

Destexhe, A., Rudolph, M., and Paré, D. High-conductance state of neocortical neurons in vivo. Nature Reviews Neuroscience, 8(9):739–751, 2007.

Fortunato, M., Gheshlaghi Azar, M., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. Noisy networks for exploration. International Conference on Learning Representations, 2017.

Fortunato, M., Gheshlaghi Azar, M., et al. Information-directed exploration for deep reinforcement learning. arXiv preprint arXiv:1812.07544, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International Conference on Machine Learning, pp. 1861–1870, 2018.

Hao, J., Yang, T., Tang, H., Bai, C., Liu, J., Meng, Z., Liu, P., and Wang, Z. Exploration in deep reinforcement learning: From single-agent to multiagent domain. IEEE Transactions on Neural Networks and Learning Systems, 2023.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Gheshlaghi Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In AAAI conference on artificial intelligence, volume 32, 2018.

Hollenstein, J., Auddy, S., Saveriano, M., Renaudo, E., and Piater, J. Action noise in off-policy deep reinforcement learning: Impact on exploration and performance. arXiv preprint arXiv:2206.03787, 2023.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In Advances in Neural Information Processing Systems, pp. 1109–1117, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., and Petersen, S. Human-level control through deep reinforcement learning. Nature, 518 (7540):529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning, pp. 1928–1937, 2016.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 16–17, 2017.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. arXiv preprint arXiv:1706.01905, 2018.

Seo, Y., Chen, L., Shin, J., and Lee, H. State entropy maximization with random encoders for efficient exploration. In Advances in Neural Information Processing Systems, volume 33, 2020.

Taiga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. On bonus-based exploration methods in the arcade learning environment. International Conference on Learning Representations, 2021.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. Exploration: A study of count-based exploration for deep reinforcement learning. Advances in neural information processing systems, 30, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In International Conference on Intelligent Robots and Systems, pp. 5026–5033, 2012.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In International Conference on Machine Learning, pp. 1995–2003, 2016.

Yuan, M., Li, B., Jin, X., and Zeng, W. Automatic intrinsic reward shaping for exploration in deep reinforcement learning. arXiv preprint arXiv:2301.10886, 2023.

Zheng, L., Chen, J., Wang, J., He, J., Hu, Y., Chen, Y., Fan, C., Gao, Y., and Zhang, C. Episodic multi-agent reinforcement learning with curiosity-driven exploration. arXiv preprint arXiv:2111.11032, 2021.

# 9. Appendix

## 9.1. NoisyQ-DDQN Algorithm

Algorithm 1 outlines the NoisyQ-DDQN algorithm, incorporating the NoisyQ variants. The algorithm seamlessly integrates the NoisyQ modifications during action selection, replay buffer management, loss computation, and parameter updates.

**Algorithm 1** NoisyQ-DDQN Algorithm

**Input**: Env Environment; $\theta$ Initial network parameters; $\theta^-$ Initial target network parameters; $\sigma$ Noise variance; $M$ Number of steps; $NB$ Replay buffer size; $NT$ Training batch size

**Output**: $Q(x, a, \theta)$ Action-value function

Choose a NoisyQ variant $\eta$ from {NoisyQ All-Q, NoisyQ Max-Q, NoisyQ Avg-Q}
**for** $t$ in $\{1, \ldots, M\}$ **do**
  Initialize state sequence $x_0 \sim$ Env
  Set $x = x_0$
  Select an action $a$ according to $max_{a \in A} Q(x, a, \theta)$
  Sample next state $y \sim P(\cdot|x, a)$, receive reward $r = R(x, a)$.
  Add transition $(x, a, r, y)$ to the replay buffer $B$
  set $x = y$.
  **if** $|B| > NB$ **then**
    Delete oldest transition from $B$
  **end if**
  **if** $t \mod NT = 0$ **then**
    Sample a minibatch of $NT$ transitions $((x_j, a_j, r_j, y_j) \sim B)$ for $j = 1, \ldots, NT$
    Sample the Gaussian noise vector according the chosen NoisyQ variant for the online network $\vec{\varepsilon} \sim \mathcal{N}$
    Construct the noisy target values using NoisyQ variants and the given $\sigma$
    Do a gradient step with the NoisyQ-DDQN loss
    **if** $t \mod N^- = 0$ **then**
      Update the target network parameters: $\theta^- = \theta$
    **end if**
  **end if**
**end for**

## 9.2. NoisyQ Entropy

The tables below show the entropy values of different models (noise energy and NRR) in three checkpoints. For each NoisyQ table, the first column on the left shows the noise level and the NRR step, respectively.

| Decay | 5000 | 10000 | 15000 |
|---|---|---|---|
| 1000 | 3.642 | 4.421 | 4.916 |
| 2000 | 3.798 | 4.727 | 4.946 |
| 5000 | 4.203 | 4.899 | 5.012 |
| 7000 | 4.117 | 4.697 | 5.186 |
| 10000 | 4.108 | 4.650 | 5.082 |

*Table 3.* Epsilon Greedy

| Noise, NRR | 5000 | 10000 | 15000 |
|---|---|---|---|
| 0.1, 500 | 4.736 | 4.982 | 5.084 |
| 0.1, 1000 | 4.982 | 5.082 | 5.055 |
| 0.1, 5000 | 4.908 | 5.003 | 5.077 |
| 0.5, 500 | 4.695 | 4.949 | 4.982 |
| 0.5, 1000 | 4.882 | 4.963 | 5.042 |
| 0.5, 5000 | 4.959 | 5.077 | 5.090 |
| 1, 500 | 4.937 | 5.073 | 5.112 |
| 1, 1000 | 5.011 | 5.064 | 4.983 |
| 1, 5000 | 4.725 | 4.898 | 4.909 |

*Table 4.* NoisyQ Method 1

| Noise, NRR | 5000 | 10000 | 15000 |
|---|---|---|---|
| 0.1, 500 | 5.014 | 5.019 | 5.042 |
| 0.1, 1000 | 4.840 | 5.020 | 5.060 |
| 0.1, 5000 | 4.616 | 4.876 | 4.896 |
| 0.5, 500 | 5.058 | 5.055 | 5.064 |
| 0.5, 1000 | 4.831 | 4.923 | 4.999 |
| 0.5, 5000 | 4.828 | 5.008 | 5.060 |
| 1, 500 | 4.743 | 4.879 | 4.999 |
| 1, 1000 | 4.964 | 5.035 | 5.038 |
| 1, 5000 | 4.791 | 4.957 | 5.007 |

*Table 5.* NoisyQ Method 2

| Noise, NRR | 5000 | 10000 | 15000 |
|---|---|---|---|
| 0.1, 500 | 4.955 | 5.015 | 5.072 |
| 0.1, 1000 | 4.959 | 5.002 | 5.053 |
| 0.1, 5000 | 4.969 | 4.983 | 4.991 |
| 0.5, 500 | 4.699 | 4.900 | 4.990 |
| 0.5, 1000 | 5.126 | 5.153 | 5.140 |
| 0.5, 5000 | 5.012 | 5.030 | 5.086 |
| 1, 500 | 4.413 | 4.780 | 4.896 |
| 1, 1000 | 4.910 | 4.946 | 4.995 |
| 1, 5000 | 4.946 | 5.081 | 5.084 |

*Table 6.* NoisyQ Method 3