Optimisation of Evolving Policy Networks with Genetic Algorithms with Alternative Noise Types for Evolutionary Reinforcement Learning

Anshali Manoharan

I. Introduction

Over the past decade, interest has surged in Evolutionary Reinforcement Learning (ERL) algorithms, a specialised subset of Reinforcement Learning (RL) methods [1]. ERL integrates RL components, such as agents or parameters, as evolving individuals within a population, optimising performance through iterative adaptation and selection using an Evolutionary Algorithm (EA). [2]. ERLs use a range of Evolutionary Algorithms (EAs), including Evolutionary Strategies (ES), Particle Swarm Optimisation (PSO), and Genetic Algorithms (GAs) [2] to enhance efficiency and effectiveness of RL algorithms.

Among these, ERL algorithms employing GAs have demonstrated particularly strong performance in complex RL tasks, often surpassing gradient-based approaches such as Deep Q Network (DQN), Asynchronous Advantage Actor Critic (A3C), and ES. This competitive advantage is underscored in several studies [3,4]. These successes highlight neuroevolution with GAs as a robust alternative, especially in structures of state spaces where gradient-based methods struggle [3].

In this context, Faycal et al. [5] demonstrated the efficacy of GAs by evolving policy network weights in RL tasks. Their work proposed two key modifications—Multi-Step Mutation (MSM) and Directed Crossover (DC)—to the baseline GA which significantly improved convergence speed and performance.

Significant progress has also been made in understanding the role of different noise types in reinforcement learning. Eberhard et al. [6] demonstrated that pink noise applied as action noise, outperformed OU noise, white noise, and other variants across various RL environments. These findings strongly advocate for pink noise as a default choice for action noise in continuous control tasks.

However, despite this growing interest in the application of alternative noise types in RL, ERL frameworks incorporating GAs continue to predominantly use Gaussian noise as the default mutation operator. [2][3][4][5]. The potential benefits of alternative noise types, such as pink noise and OU noise, in the mutation process are unexplored.

Additionally, while pink noise has demonstrated effectiveness as action noise, its application as noise within continuous input spaces remains unexamined, despite its growing recognition as a realistic representation of typical environmental variability [7]. Pink noise, when applied as input space noise in GAs to test the generalisability of the individuals has not been systematically investigated.

This study uses the GA proposed by Faycal et al., GA-MSM, and GA-MSM-P, a novel modification to GA-MSM as the primary frameworks for investigation, and addresses the aforementioned gaps framed as the following two research questions: (i) Does the choice of noise type affect the mutation process of GAs, particularly non-Gaussian noise types (e.g., OU, pink)? If so, how does different types of noise influence average fitness across generations and convergence performance, compared to the standard Gaussian noise? (ii) How does injecting pink noise into the input space, rather than directly into the action space, affect average fitness across generations and convergence performance in GAs?

This study presents the following contributions: (i) Recommendations of noise types for mutation operators in GAs based on empirical results. (ii) Examination and analysis of the impact of injecting pink noise into the input space on convergence in GAs. (iii) A novel modification to the GA-MSM algorithm, GA-MSM-P (Genetic Algorithm with Multi-Step Mutation and Preservation). Empirical results shows that it performs statistically significantly better than GA-MSM.

II. Background

Genetic Algorithms (GAs) is a type of EA inspired by Darwin's "survival of the fittest" theory [8]. It evolve populations of potential solutions using the following operators: selection, crossover, and mutation. A fitness function guides the selection of high-quality individuals for generating the next generation – these are usually

known as the elites of the population. Crossover is performed, where elites are combined to produce offspring for the next generation. Mutation is then applied on the offspring using Gaussian noise to maintain diversity in the population [8]. GAs are known to be highly parallelisable excelling in various applications such as scheduling, bioinformatics, and robotics [9].

A study by Faycal et al introduces an GA named GA-MSM, a baseline GA with a novel modification, Multi-Step Mutation (GA-MSM). GA-MSM is an GA which generates populations of individuals, each representing a set of policy network weights, and evaluates their fitness based on the policy network's performance in the environment. Evaluation on the FrozenLake environment shows that GA-MSM performs significantly better over the baseline GA, demonstrating convergence in fewer generations and at much higher average fitness levels. This study provides incentive for further experimentation, because this study also uses the default Gaussian noise as its mutation operator. Furthermore, this GA's heavy reliance on mutation compared to the standard GA makes it well-suited for analysing the behaviours of different noise types as mutation operators.

Different noise types have been explored to improve exploration in RL, particularly in continuous action spaces. The aforementioned study by Eberhard et al. [6] also points out the traditional but ineffective sampling of actions from uncorrelated Gaussian distributions, which does not provide sufficient exploration for many RL tasks. The study discusses that pink noise's balance between local and global exploration leads to uniform state space coverage. The detailed evaluation of noise types and empirical results sets a solid understanding on the nature and superior capabilities of pink noise as default action noise in RL environments, highlighting its potential as a mutation operator, and possibly as input space noise, given its ability to simulate environmental fluctuations as well. [7].

III. METHODS

This section describes the GAs used to address the goals of the study, and the experimental set up used throughout the study.

A. Algorithms

A.1 Genetic Algorithm with Multi-Step Mutation

The GA-MSM algorithm has one key distinction from the standard GA - the mutation step. The standard GA executes a single mutation step per individual before proceeding to the next generation, whereas GA-MSM applies multiple iterations of mutation (*n* additional times) on the current population. If the resulting mutated population demonstrates superior environment-based fitness, it replaces the current population and proceeds to the next generation. Otherwise, crossover and one round of mutation are applied to the current population. This time, the fitness would be calculated using dissimilarity from the underperforming mutated population as the fitness measure instead. The dissimilar elites are used to crossover and mutate to create the next generation. This ensures directed diversity in the individuals [5]. The pseudocode for GA-MSM is outlined in Algorithm 1, Section B, Appendix.

A.2 Genetic Algorithm with Multi-Step Mutation and Preservation

A novel version of GA-MSM was experimented in this study. This novelty stems from an observation in GA-MSM, where every time the mutated population performs worser than the current population, the newly created population's fitness relies solely on how different the new population is from the mutated population. While this introduces directed diversity to the new population, this measure discards the accumulated evolutionary progress in earlier generations, as shown in Figure 6, Section A, Appendix.

To counter this while still maintaining the diversity measure, the following modifications were made to GA-MSM: (i) Top 10% of the dissimilar individuals were preserved in the population. (ii) Original mutated population's elites were used to create the remainder of the new population with mutation and crossover, instead of the dissimilar elites. This version of the algorithm is outlined in the Algorithm 2, Section B, Appendix.

B. Experiments

In addition to the experiments to address the goals of this study, a performance comparison between GA-MSM and GA-MSM-P is also implemented.

B.1 GA-MSM vs. GA-MSM-P

To evaluate the performance of the modification, 10 runs are conducted for both GAs on the FrozenLake environment, and the medians of the average population fitness scores across the 10 runs are analysed using the pairwise Mann-Whitney U Test for statistical significance. A

visual representation of this methodology is provided in Figure 7, Section A, Appendix. Hyperparameters are set consistent to the original framework by Faycal et al., as detailed in Table 3, Section C , Appendix.

B.2 Evaluation of Alternative Noise Types as Mutation Operators

To assess the performance of Gaussian, pink, and OU noise as mutation operators, the GAs are evaluated over 10 runs on the Frozen Lake environment. Due to its simplicity and small scale, Frozen Lake is well-suited for isolating the effects of different noise types, allowing for a clearer analysis of algorithm behaviour under various mutation operators. The mutation operators function by adding noise directly to the neural network weights, which are then clipped to remain within limits calculated using Glorot uniform initialisation. For each mutation operator, 10 runs are performed per GA, with median scores calculated across 200 generations. Pairwise Mann-Whitney U tests are conducted to evaluate the statistical significance of differences between Gaussian and pink, Gaussian and OU, and pink and OU noise. Further details on environment configuration and implementation of mutation operators for this experiment are in Section D and E of Appendix, respectively.

B.3 Effect of State Space Pink Noise Injection on GA Convergence

The second experiment examines the impact of pink noise injection into the input space on the GAs' convergence performance in the CartPole environment. Pink noise is added with constant scale $\sigma \in \{1.0, 0.5, 0.1\}$ throughout evaluation. For each scale, five runs of the GA are conducted, with fitness assessed over 10 episodes per individual. An additional five runs without noise injection are included for comparison. Median scores of average population fitness across generations are reported across 5 runs. Further details for pink noise injection into input space are outlined in Section F, Appendix.

C. Performance Measures

Performance is evaluated using three metrics: (i) the number of episodes required to produce optimal agents, (ii) the number of generations before convergence is reached, and (iii) the average population fitness at convergence. Convergence criterion is defined as follows: for FrozenLake, when the change in average population fitness over 10 generations is < 0.03; for CartPole, when the change is < 5.

IV. RESULTS

In this section, results of the experiments are shown. Firstly, performance comparison between GA-MSM and GA-MSM-P are shown in Figure 1.

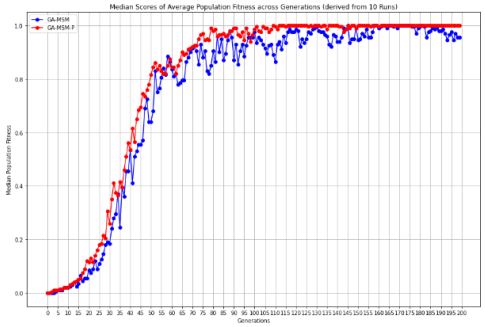

Figure 1. Comparison of GA-MSM and GA-MSM-P using10 runs on Frozen Lake Environment.

GA-MSM-P demonstrated statistically significant performance improvements over GA-MSM ($p < 0.01$). Subsequently, both algorithms were evaluated under varying noise mutation operators, with results presented in Figures 2 and 3.

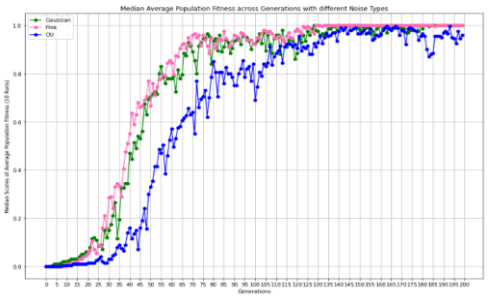

Figure 2. Effects of Gaussian, OU and pink noise as mutation operators on GA-MSM.

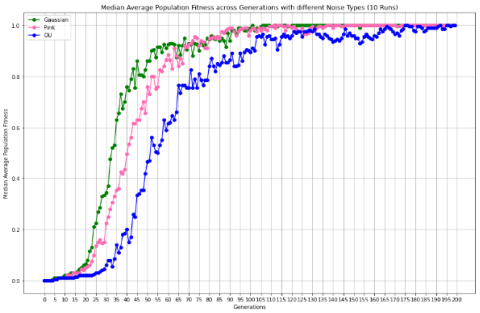

Figure 3. Effects of Gaussian, OU and pink noise on mutation operators of GA-MSM-P

The results of the Mann-Whitney U tests indicate that Gaussian and pink noise exhibited similar performance in both GAs, showing no statistically significant difference. However, OU noise demonstrated statistically significant underperformance relative to both Gaussian and pink noise ($p<0.01$). Table 1. shows performance metrics for the noise types.

Table 1. Performance Metrics for each Noise Type

Noise Type		Gaussian	Pink	OU
No. episodes to produce optimal agents	**GA-MSM**	13500	12900	14000
	GA-MSM-P	9100	10000	11500
Number of generations taken to converge	**GA-MSM**	135	129	140
	GA-MSM-P	91	100	115
Average population fitness at convergence	**GA-MSM**	0.98	1.0	0.98
	GA-MSM-P	0.98	0.98	0.96

The second experiment's results for different scales of pink noise injection into input space for GA-MSM and GA-MSM-P in the Cartpole environment-v0 is shown in Figure 4 and 5.

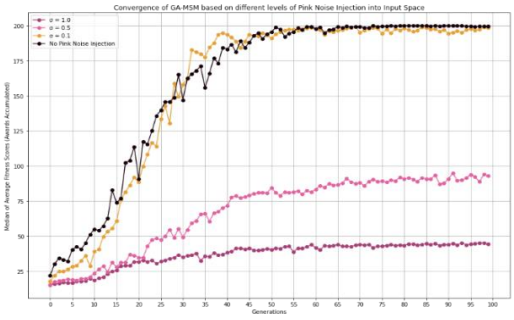

Figure 4. Effect of different levels of pink noise injection on continuous input space for GA-MSM.

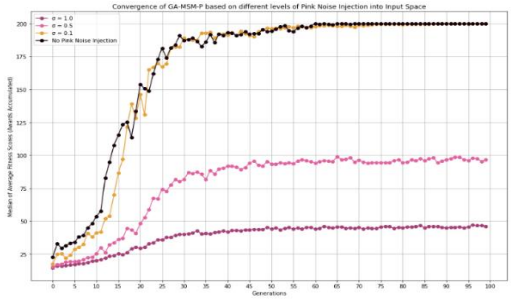

Figure 5. Effect of different levels of pink noise on continuous input space for GA-MSM-P

Table 2. Performance Metrics for each Pink Noise Scale (σ)

σ value		1.0	0.5	0.1	0
No. episodes to produce optimal agents	**GA-MSM**	16000	51000	51000	55000
	GA-MSM-P	27000	45000	38000	38000
Number of generations taken to converged	**GA-MSM**	16	51	51	55
	GA-MSM-P	27	45	38	38
Average population fitness at convergence	**GA-MSM**	28.56	80.95	193.89	195.57
	GA-MSM-P	37.73	94.10	192.12	192.13

V. Discussion

The study addresses the posed research questions and provides the following key insights. Empirical results presented in Table 1. and in statistical tests demonstrate that the choice of noise type for mutation operators significantly influences the number of generations taken to converge. Pink noise performs comparably to Gaussian noise, with no significant difference in their effectiveness as mutation operators for both GAs, whereas OU noise leads to slower convergence. This could be due to OU's tendency to produce large perturbations [6] which lead to excessive mutation of individuals. Additionally, it is observed that the type of noise mutation has a negligible impact on the average population fitness at convergence, as shown in Table 1. Conclusively, the findings suggest that pink noise serves as a viable alternative to Gaussian noise for mutation in GAs. Experiments on pink noise injection into the input space reveal that the GAs generalise well under minor perturbations (scale = 0.1), as seen in Table 2. However, higher noise scales (0.5, 1.0) reduce average fitness at convergence heavily, indicating that excessive pink noise disrupts evolutionary progress due to distorted state space representations.

Additionally, this study presents a novel modification, GA-MSM-P which enhances convergence speed significantly by preserving evolutionary progress. Figures 1, 4, and 5 highlight clear promising trends of its progressive fitness increments across generations, outperforming GA-MSM, in convergence speed. GA-MSM-P reduces the episodes required to produce optimal agents (Tables 1 and 2) while maintaining comparable average fitness at convergence, addressing the typical limitation of slow convergence in GAs [1] and enhancing computational efficiency. Lastly, this study highlights promising directions for future work. Due to computational constraints, GA-MSM-P's performance in more complex environments remains unexplored and warrants further investigation to generalise its applicability. Additionally, the observed trends in noise types for mutation operators, particularly the comparable performance of pink and Gaussian noise can be further investigated whether it is environment dependent or if it generalises across environments. Furthermore, GAs behaviour on alternative decay factors for the noise mutation operators, beyond the default decaying factor used in this study can be explored.

References

[1] Y. Song et al., 'Reinforcement learning-assisted evolutionary algorithm: A survey and research opportunities', Swarm and Evolutionary Computation, vol. 86, p. 101517, 2024.

[2] Q. Zhu et al., 'A survey on evolutionary reinforcement learning algorithms', Neurocomputing, vol. 556, p. 126628, 2023.

[3] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, 'Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning', arXiv preprint arXiv:1712. 06567, 2017.

[4] A. Seth, A. Nikou, and M. Daoutis, 'A scalable species-based genetic algorithm for reinforcement learning problems', The Knowledge Engineering Review, vol. 37, p. e9, 2022.

[5] T. Faycal and C. Zito, 'Direct mutation and crossover in genetic algorithms applied to reinforcement learning tasks', arXiv preprint arXiv:2201. 04815, 2022.

[6] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius, 'Pink noise is all you need: Colored noise exploration in deep reinforcement learning', in The Eleventh International Conference on Learning Representations, 2023.

[7] M. Grove, J. M. Borg, and F. Polack, 'Coloured noise time series as appropriate models for environmental variation in artificial evolutionary systems', in Artificial Life Conference Proceedings 32, 2020, pp. 292–299.

[8] S. Katoch, S. S. Chauhan, and V. Kumar, 'A review on genetic algorithm: past, present, and future', Multimedia tools and applications, vol. 80, pp. 8091–8126, 2021.

[9] T. Alam, S. Qamar, A. Dixit, and M. Benaida, 'Genetic algorithm: Reviews, implementations, and applications', arXiv preprint arXiv:2007. 12673, 2020.

[10] S. Sinha, A. Mandlekar, and A. Garg, 'S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics', in Conference on Robot Learning, 2022, pp. 907–917.

APPENDICES

A. Experimental Design & Observations

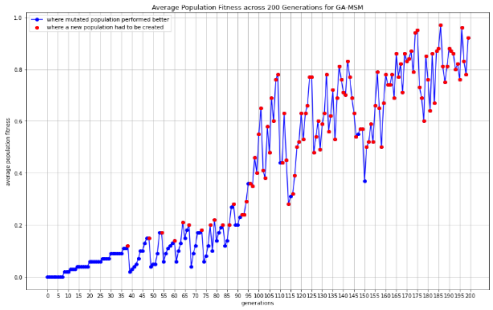

Figure 6. Graph highlighting a consistent observation while running GA-MSM – its tendency to discard evolutionary progress with each new population created in earlier generations.

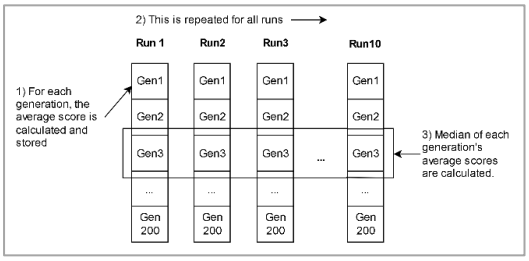

Figure 7. This method was executed for each noise type, the median scores were used for testing statistical significance.

B. Algorithms and Hyperparameters used

Algorithm 1: GA-MSM

$n \leftarrow$ number of additional mutations
$p \leftarrow$ *generate_population()*
while *not converged* **do**
  $f_{current} \leftarrow$ *compute_fitness(p)*
  $e \leftarrow$ elite performers
  $p_{mutated} \leftarrow e$
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    $p_{mutated} \leftarrow$ *mutate(crossover($p_{mutated}$))*
  end
  $f_{mutated} \leftarrow$ *compute_fitness($p_{mutated}$)*
  if $f_{mutated} \geq f_{current}$ **then**
    $p \leftarrow p_{mutated}$
  else
    $p_{new} \leftarrow$ *mutate(crossover(e))*
    $f_{new} \leftarrow$ compute_fitness(p_{new}, $p_{mutated}$)
    $e_{new} \leftarrow$ select elite according to f_{new}
    $p \leftarrow$ *mutate(crossover(p_{new}))*
  end
end

Algorithm 2: GA-MSM-P

$n \leftarrow$ number of additional mutations
$p \leftarrow$ *generate_population()*
while *not converged* **do**
  $f_{current} \leftarrow$ *compute_fitness(p)*
  $e \leftarrow$ elite performers
  $p_{mutated} \leftarrow e$
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    $p_{mutated} \leftarrow$ *mutate(crossover($p_{mutated}$))*
  end
  $f_{mutated} \leftarrow$ *compute_fitness($p_{mutated}$)*
  if $f_{mutated} \geq f_{current}$ **then**
    $p \leftarrow p_{mutated}$
  else
    $p_{new} \leftarrow$ *mutate(crossover(e))*
    $f_{new} \leftarrow$ compute_fitness(p_{new}, $p_{mutated}$)

```
        e_new ← select elite according to f_new
        p ← e_new
        p ← mutate(crossover(e))
    end
end
```

C. Configuration for GA-MSM and GA-MSM-P

Table 3. Structure & Hyper Parameters (based on original framework)

GA Structure	GA Details	Extra Notes
Policy Network Type (per individual)	Feed-Forward	-
Network Shape	16-10-10-4	16 input states, 4 available actions, 2 hidden layers
Total Parameters	300	Number of weights in the network
Selection Operator	Elitism	Top 10% of performers who accumulated most rewards from environment
Crossover Operator	Uniform	Randomly choosing genes from two parents from elite population
Mutation Operator	Gaussian (Default)/ Pink/OU	Generated as same shape as weight matrix, with decay factor set as *1/(1+ generation number)* to manage the scale of noise.
MSM Steps	10	Number of extra times elite population is crossed over and mutated
Population Number	100	Number of individuals per generation
Activation Functions	ReLu & Softmax	ReLu for first three layers, Softmax for final output layer
Fitness Evaluation	Performance within one episode in the environment	The total reward accumulated within one episode in FrozenLake environment.
Decaying Factor	*1/(1+ generation number)*	Original framework for GA-MSM does not specify a specific decaying factor, therefore a default decaying factor of 1/x is used. It gradually reduces magnitude of mutations as the GA progresses.

D. Environment Configuration

Table 4. FrozenLake Environment Configuration

Configuration	Value	Justification
Version	1	-
Map Size	4 x 4	Default setting used.
is_slippery	False	This option increases stochasticity, mimicking real-world unpredictability, however, non-slippery settings isolate the effects of different noise types on performance and convergence, therefore is_slippery = False.
Number of episodes run per Individual	1	1 episode is carried out per individual, due to the non-stochastic nature of the environment.
Reward	Standard	As specified in the Gymnasium Documentation

Table 5. CartPole Environment Configuration

Parameter	Value	Justification
Version	0	Uses 200 timesteps per episode, less use of computational resources
Initial state	Random	To assess the individual's ability to generalise to random events within the environment.
Number of episodes run per Individual	10	10 episodes per individual to assess fitness, a balance between computational constraints and providing a fair assessment of the individual's performance in the environment.
Reward	Standard	As specified in the Gymnasium Documentation

E. Mutation Operators using Different Noise Types – Formal Implementation

Mutation operators performed noise injection into the individuals in the following manner:

Let $\omega = \{\omega_1, \omega_2, \omega_3, \dots\} \in \mathbb{R}^n$ represent the flattened vector of policy network weights, where n is the total number of weights in the network.

A noise vector *x*, according to the noise type is sampled. In the case of Gaussian, $x_{Gaussian}$, is sampled independently for each weight. Temporally correlated noise like OU and pink have noise vectors (x_{pink} and x_{OU}) that are generated based on their previous samples.

The noise vector is added to the vector of policy network weights, $\hat{\omega} = \omega + d(x)$ where *x* can be of the specified noise type, and function *d( )* controls the noise vector's amplitude, which is controlled by the decaying factor used in GA-MSM (check Table 1.) The noise's amplitude is supposed to decay over generations, as specified by the original framework for GA-MSM.

The noisy vector of policy network weights $\hat{\omega}$ is reshaped back to its original layer-wise structure, $\hat{\omega} = \{\hat{\omega}^1, \hat{\omega}^2, \hat{\omega}^3 \dots \hat{\omega}^L\}$ where L is the number of layers and $\hat{\omega}^1$ represents the noisy weights for layer 1.

Finally, the noisy weights are clipped according to the Glorot uniform distribution's bounds: $clip(\hat{\omega}^L, -\Delta, \Delta)$ where $\Delta = \sqrt{(\frac{6}{n_{in}+n_{out}})}$ where n_{in} and n_{out} are the number of inputs and outputs respectively for layer L. The final noisy parameter set for the policy network is with weights adjusted layer-wise and is finally flattened to be fed back into the GA pipeline.

This process ensures both diversity (via noise injection) and stability after noise mutation (via Glorot clipping).

The GitHub libraries/projects used for generating pink and OU noise are linked in the following table.

Noise Type	**Library/Code adaption**
OU Noise	GitHub - soeren-kirchner/project2-reacher
Pink Noise	GitHub - felixpatzelt/colorednoise: Python package to generate Gaussian (1/f)**beta noise (e.g. pink noise)

F. Continuous Input Space Pink Noise Injection – Formal Implementation

Since pink noise exhibits temporal correlation, it cannot be independently sampled at each time step for injection into the continuous input space. Instead, the noise samples must be pre-generated before the start of the episode.

These samples should match the number of time steps in the environment and align with the dimensionality of the state space. Consequently, the noise array will have the shape *(episode length, state_space_size)* and will be referenced at each time step for noise injection.

Noise is applied to the state space, using the simple noise augmentation technique adopted by Simha et al. for zero-mean Gaussian noise for state transformation [10], where noise is simply added to the continuous input space.

The noisy state space array is then clipped according to the applicable ranges specified in the Gymnasium Documentation for the state space values like pole angle and cart position, before being fed into the policy network.