IMPERIAL

ExNEAT - Separable network through coevolution of partially-dependent sub-modules

Final report

Author

Mengjie Zhang

CID: 02041373

Supervised by

Dr Daniel F. M. Goodman

Dr. Marcus Ghosh

Second Marker

Dr. Giunchiglia

A Thesis submitted in fulfillment of requirements for the degree of
Master of Engineering in Electrical and Electronic Engineering

Department of Electrical and Electronic Engineering
Imperial College London

2025

Abstract

Neuroevolution (NE) is an alternative method for training artificial neural networks. NE based task solving works by initializing a large number of randomly generated networks and testing each individual network to compute a fitness score. This fitness score is used to select the most promising networks and use them to reproduce the next generation of networks. Applying this process iteratively allows convergence towards an optimal solution to a task. *Neuroevolution of argumented topologie* (NEAT) [1] is a popular NE algorithm that have been commonly used for evolving neural networks. This thesis presents a novel neuroevolution algorithm named *Externally-integrated ModularNEAT* or *ExNEAT* that was built based on the *NEAT* algorithm. In this project, we have developed a complete *ExNEAT* pipeline and solved a selection of 3 navigation and control tasks using *ExNEAT*. We have addressed some of the previous limitations of the original NEAT paper and presented the experimental results which shows that *ExNEAT* exhibits significant improvements compared to the original NEAT in tasks that involved navigation and control in Cartesian spaces. We have also demonstrated that networks trained with *ExNEAT* are more robust against environmental uncertainty and are more efficient in terms of connectivity and neural structures.

Declaration of Originality

I hereby declare that the work presented in this thesis is my own unless otherwise stated. To the best of my knowledge the work is original and ideas developed in collaboration with others have been appropriately referenced.

A small portion of the development of the project have employed the use of LLMs to speed up the design process. No external assistance has been used in the writing process of the report apart from minor grammar corrections.

Copyright Declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.

Acknowledgments

The early ideas and experiments of this project were developed collaboratively with the project supervisor(s) and the interim report was reviewed by the supervisor(s).

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Overview

Intelligent robots have existed from as early as the 1960s and since then, the development of artificial intelligence in the field of robotics has undergone a surge in popularity [2]. With the advent of more sophisticated techniques in applied machine learning and especially, the advancement in deep learning, more versatile robotic systems were developed. Because ultimately, robotic systems are used to assist humans in their accustomed environments. Thus robots that exhibit natural motions and actions can be more easily deployed in their target settings. For that reason and many others, robotic systems with artificial limbs such as legs and arms are particularly popular trend among researchers. Robots that mimics the biological structure of living creatures, such as quadruped robots and humanoids remain superior to wheeled robots due to their ability to manoeuvre in more complex terrain and thus perform more complex actions [3]–[5]. As the complexity of such systems

groups, the difficulty of stable navigation while performing tasks becomes challenge. Moreover, the current trend in artificial intelligence often demands a robot to utilize signals from difference sources and perform a number of tasks simultaneously. Hence, the development of control and navigation have received a high attention [6], [7]. And in many research studies, the development of any biologically inspired robots often requires an excellent control system that deals with static and dynamic spatial information with good bandwidth. Such control system is often the foundation of an intelligent robotic system and any extra features such as vision or language processing will need to be built upon that. This thesis will describe some of the most widely used techniques in robot learning and then introduce a novel approach developed to solve some representative robot navigation and control problems. The first two problem involves static, sequential decision making of a single-body agent in 1D and 2D space while the third problem requires comprehensive control and navigation of multi-body, dynamic agent in 3D space. The project utilizes the NEAT-Python library and aims to build a complete pipeline for the implementation of *ExNEAT*. We will be running thoroughly testing on both NEAT and *ExNEAT* agents and using comprehensive results data to show how much improvement our new algorithm can offer.

1.2 Background and related work

1.2.1 Reinforcement learning

Reinforcement learning (RL) [8] has always been the popular center for robotics learning. So far, many methods have been introduced to continuously perfect this subfield of deep-learning. In traditional machine learning, goal of learning is typically to approximate a certain function that can represent and generate a pattern or trend. The baseline for assessing any model's performance is based on the error of approximation. Reinforcement learning differs from traditional machine learning in the way that performance measurement is done by recording a cumulative score, determined by the degree of completion of a certain task assigned to an intelligent system (usually referred to as an agent in RL). The overall process of reinforcement learning is typically described as a Markov decision process within a certain Markov chain. Where at each time step, the agent has the choice of selecting a range of possible actions (called the policy of an agent) and each action leads to a different state in the following time step. As an agent explores difference choices of states, it may be rewarded for taking a right action, or be penalized for taking a bad action. The goal is to learn a policy that maximizes the probability of obtaining the highest reward until

the end of exploration [9], [10]. Reinforcement learning is unique in the sense that it possesses some resemblance to natural biological learning. For example, the animals inside a circus may get food for good performance while being punished for each failure, and over time, the animal learns how to be a good performer.

Reinforcement learning agents are trained in a similar manner such that an agent is put into a test field (called an environment) where it explores and learns. A reinforcement learning process usually involves 2 stages, exploration: where the agent randomly samples the action space to learn the good and bad choices, and exploitation: where the agent uses information gathered from the exploration stage to perform tasks in both seen and unseen environments. Deep reinforcement learning (DRL) [8] is the combination of RL with deep neural networks. In DRL, the memorization process in encoded in a neural network and backpropagation is used to update the policy learned by that network. There have been a great number of RL benchmark environments such as the single and double inverted pendulum, the mountain bike, the swimmer and the walker task. In practice, any dynamic or sequential decision making tasks can be used as an RL training environment, for example, learning to open a door or to playing chess. Because of this, reinforcement learning agents has a wide range of applications ranging from daily manual tasks to specialized operations. Some of the popular RL algorithms includes Q-learning [11], SARSA [12] and stable baseline [13].

While most approaches had safely anchored the training techniques to the traditional neural network gradient descent (GD), i.e., using backpropagation (BP). Despite the initial success of using gradient-based optimization, there are likely to be alternative methods that are more suitable for training deep learning models that are designed specifically for reinforcement learning purposes. Furthermore, GD only optimizes the weights of the model while disregarding the options where altering model structure could shed light on better solutions with possibly a much lighter model size **24**. In terms of the main goal of the project, the trial-and-error nature of reinforcement learning makes it suitable for training a neuroevolution (NE) agent. Since NE learns the perfect policy to a problem without explicitly computing a relationship between weights and output, but instead relies on using the reward of each trial to estimate the best solution.

1.2.2 Evolutionary algorithms and neuroevolution

The earliest examples of applying evolutionary algorithm for the optimization of parameter models can be dated back to the early 1960s. The initial inspiration was large biologically inspired and the general idea was that: if natural selection has worked so well in evolving intelligent organism

(such as humans) in an eco-system, then would it be possible to replicate the end-results of this evolution using a much faster computing algorithm? Some of the early methods have come up with the solution that simple parameter-based functions or finite state machines can be evolved in such a way as to discover better solutions. Then digital forms of "organisms" can be evolved using mutation and crossover to eventually discover an optimal solution to a problem. Hence the term *evolution strategy* (ES) [14], [15] was named and it refers to any attempts to perform parameter optimization that involves selection and reproduction of genomes in a population. The concept of *Evolution programming* (EP) [16], [17] was another branch of evolutionary algorithm. The main difference between ES and EP is that in EP, there is no *crossover* stage as in ES and the only means to reproduce offspring is by mutating the individuals. In short, there is no such concept of *parents* or *children* as in a real evolution system. That said, both ES and EP focused on explicitly modifying the parameters by simply adding a random perturbation determined by its variance. A third type of evolutionary algorithm is called genetic algorithms (GA) [18]–[20]. This type of algorithm inherits most of the common features from the previous 2 categories, namely initializing populations, crossover, mutation and selections are all included as before. The main advancement for GA is to used of *genes*. Unlike ES or EP, where each parameter is the real number representation of its own, GA represents each parameter by encoding each number in the form of a gene. The genes from all the parameters of a solution function form the *genome* or *genotype* of that function. This way, evolutionary information can be implicitly and compactly represented by some finite dimensional vectors. Therefore, changing the genes of a function is equivalent to changing the parameters. One common scheme or representation for the genes is the *binary string representation*. And when it comes to mutation or crossover, we only need to perform operation on these binary genes instead of directly on the numbers.

In summary, an evolutionary algorithm is any optimization method based on some pre-defined evaluation-and-selection algorithm which mathematically emulates the natural selection. It begins with creating a population of individuals in an environment with specific survival constraints to enforce an evolution behavior to form different species. The process typically involves:

1. Initialize a population, p_0 which represents the initial solution space. Each member in p_0 is a potential solution and is called an *individual* (sometimes referred to as a *genome*).

2. Assess how well these solutions are at accomplishing a certain task. A fitness function, θ is used to assess each individual hypothesis in p. The fitness evaluation strategy depends on the nature of the tasks and is often customized.

3. Based on the fitness of the individuals, select the top $M\%(M < 100)$ performing individuals for reproduction.

4. (only for ES and GA) Carry out reproduction by crossover 2 best performing solutions to produce a child network.

5. Mutate the child individuals for genetic diversity. This is done by adding a random noise to the genes.

6. Over time, a new population, p_1 is produced by the child solutions.

7. The steps 2-6 are repeated until a solution with acceptable fitness is found. Or else in the event that all the hypothesis fails the fitness test (referred to as total extinction) then the population will be reset.

Neuroevolution (or NE in short) [21]–[23], is the application of genetic algorithms on artificial neural networks (ANNs) in an attempt to discover more powerful solution networks throughout the training process. ANNs are essentially a mathematical function consists of parameters called weights (w). A neuron is itself the simplest form of a network and is expressed as: $a_i = \theta(\mathbf{w}\mathbf{x}^T) = \theta w_0 x_0 + w_1 x_1 + ... + w_i x_i$ where x is the input vector, θ is called an activation function, it is a function that is applied to the dot product and its purpose is usually to clip the output to a certain range. o_i is the output. An ANN is formed from multiple neurons connected together. This means that the output o_i of a neuron is used as the input x_i of another neuron. Hence, any genetic algorithm used to optimize parameters in a function can be used to optimize a neural network - the weights can be encoded as genes as usual, the connectivity between neurons can be represented in the same way and the activation function can be selected using an integer, e.g. 0 for f_0 and 1 for f_1, which is also encode-able. In a neuroevolution process, the neural networks are evolved to generate more optimal offspring and performing this repeatedly will theoretically converge to a desirable solution (Figure 1.1). The use of genomes in genetic algorithms provides a perfect sample space for efficient search of both the optimal weights and neural architectures. The neuroevolution approach can be viewed as a biologically inspired alternative to backpropagation.

The earliest form of NE pipelines focused only on optimization of the network's neurons, i.e. weights while keeping the connectivity pattern fixed [24]–[30]. Later improvements have made it possible to evolve both the weights and topologies [1], [31]–[34]. The NE method has gained a large popularity in areas of reinforcement learning [33], [35], [36] where a programmable agent is deployed in a certain environment to solve tasks that require progressive adjustment to the

agent's model through time to learn a policy of Markov decision process (MDP). The use of evolutionary algorithms becomes advantageous when the problem or task to be solved is difficult to be characterized completely or concisely [22], [23], [34]. For instance, an environment may impose an infinite possible states on the agent or it may be difficult to define an analytical relationship between the network parameters and output loss, etc. [21], [37]. Genetic algorithms have been proven to work especially well in mobile robot control and navigation tasks [34], [36], [38]–[41].

1.2.3 NEAT

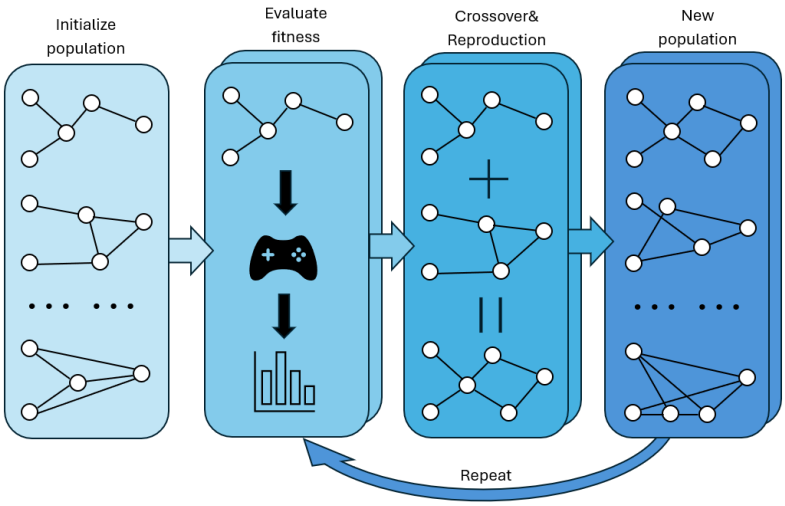


Figure 1.1: Overview of a neuroevolution algorithm

Neuroevolution through argumented topologies or *NEAT* in short, is one of the most popular neuroevolution techniques to date [8]. Like any genetic algorithms, this approach assigns a genetic encoding or genome to each individual network and performs crossover to generate offspring as usual. What was done differently to the evolutionary methods in the past was that the algorithm assigns a historical marking to each neuron in the network and a genetic distance is used to group genetically similar models in the same group in a process called speciation. This means that how many times a neuron or network has been crossovered and mutated is precisely recorded. This offers two important advantages: (1) Each genome can be identified and classified more efficiently. Which prevents good and bad solutions being mixed together. (2) Every species will be given enough generation to reach its best performance before competing with other species. This prevents the algorithm from accidentally discarding a potential genome that could become a best solution later on.

However, one big drawback of the NEAT algorithm is that it has little constraint on the

complexity of the model. Hence, it is likely that a NEAT-based agent evolves an overcomplicated architecture which slows down computation time and waste resources. With ExNEAT, the solution can be improved by dividing a network into simplified structures that is configurable and resource efficient.

1.2.4 HyperNEAT

Although the NEAT algorithm has been recognized as one of the industrial standards for NE based pipelines, a notable improvement to the NEAT method is described in the work published by Stanley et al. 2009 [42]. In which they have proposed a novel way to evolve large scale networks through implicit genetic encoding. The idea was the use another, separate NEAT neural network. This network produces a mapping of connection weights based on their spatial coordinates. Since this mapping network described completely the structure of the neural network, the NEAT process can be ascended from ”evolving a population of networks” to ”evolving a population of networks that generates the networks”. This form of genetic representation of especially compact and efficient as it allows for topology generalization up to an infinite resolution, making it feasible for a single trained genomes to be deployed to a much more complex system than it was originally trained for without any further adjustment.

1.3 Modularity in neuroevolution

1.3.1 SANE

There have also been models that took advantage of coevolving a multi-network system that consists of many smaller sub-networks that are jointly evolved together. Such systems employ a symbiotic relationship between the sub-networks and offer a more efficient way for solution search. The Symbiotic adaptive neuroevolution (SANE) is one of the earliest methods in evolving network that is composed of smaller sub-sections. As described in the original paper [43], instead of evolving populations of neural networks, a population of neurons are evolved. The neurons act as tiny building blocks for a larger network. For each training iteration, a set of neurons are selected from the population and combined into a larger substrate network. The fitness score is then assigned to each neuron and the neurons with highest fitness are selected for reproduction.

1.3.2 ModularNEAT

The idea of incorporating modules into the evolution process of a neural network had been first introduced by the ModularNEAT paper (Stanley et al. 2004) [43]. In which it described the method of evolving species of modules using a population of blueprint networks. The modules are later combined using the blueprints into a larger network. The idea is similar to that of HyperNEAT **16**, but instead a pool of modules were used to produce a pattern or symmetry in a network.

2

Requirements

Contents

2.1 Project plan and objectives

2.1.1 General aim

The main target of the project is to investigate and demonstrate the capability of *ExNEAT* in solving sequential reinforcement learning tasks that involves multi-modal sensing and navigation. The first phase of the project will be used to design a simple prototype of *ExNEAT* in order to prove its initial advantages. In the second phase of the project, we will be focusing on expanding the task complexity by adding more degree of freedom. The final phase of the project will aim to demonstrate that *ExNEAT* can not only solve simple simulated tasks but also complete tasks that are more general and involves real physics and dynamics. The first 2 phase is essential in proving the versatility of *ExNEAT* in navigation, while the final phase will attempt to push the limit of *ExNEAT* and carry out further investigation that relates to its characteristics and performance.

Throughout the design process, we will also be improving the current *ExNEAT* model to add more features to its evolution process.

2.1.2 Develop the complete *ExNEAT* algorithm

One major goal of this project us to build a novel algorithm upon the software packages of NEAT-Python so that it can be used seamlessly in this new application. A base algorithm of *ExNEAT* evolves identical modules across the entire network to achieve symmetry. The individual module blocks should evolve independently and each modules also functions independently, meaning they not yet share any connections between them. As we move onto the more complex tasks, we will be adding the ability for all the modules to evolve using a custom configuration. This will allow the agent to explore a solution space with a higher degree of freedom. Because other configurations may also be needed to complete tasks that are more challenging than maze solving. This also opens up greater possibilities for different evolution trials. Moreover, as the project transits into mid-stage, we will also add inter-connections between modules and make these connections evolvable. This will allow the agent to explore the inter-dependency between the co-evolved modules and will allow for more stable control of complex systems. One important objective of this project is to disambiguate the relationship between multi-modal learning and neuron development, hence each modulized network needs the intercommunication to function as a whole network. Moreover, evolving the topology module-wise may lead to more powerful solutions.

This part of the project should be completed before the end of Spring term. This is a realistic goal as it builds upon the knowledge gathered and initial results achieved so far, therefore it should not add too much burden on top of the other coursework and examination preparation tasks.

2.1.3 The maze navigation problem

In the first phase of the project, the goal should be to demonstrate the basic functionality of *ExNEAT* in simple game environment. *ExNEAT* should be expected to easily achieve superior performance compared to NEAT and establish an initial benchmark of how *ExNEAT* can serve as a better alternative to NEAT. A thorough evaluation of *ExNEAT*'s performance in the maze solving experiment will then be used to set a realistic milestone for the next phase of the project.

2.1.4 The predator and prey problem

The second task should be the ones that is difficult to solve using solutions provided using the original *NEAT* algorithm. We introduce the *predator and prey* problem, which involves training of multiple agents with different task goals. The prey agents, is a group of population to be initialized within a confined area. The prey agents need to gather food to survive; the predator agents has the sole aim of capture as many prey agents as they can in a limited time. This experiment is a better imitation of the competitive natural survival environment and hence the use of NEAT and it's improved models would help us to answer questions about the relationships between the nature's evolution process and machine learning. And with the aid of *ExNEAT*, this would surely open up better alternative solutions during the training of reinforcement learning agents. As have been previously shown, the predator-vs-prey task is hardly solvable using the NEAT algorithm. It is expected that *ExNEAT* will be able to solve the task with excellent performance. A careful comparison will be drawn to highlight the strength of *ExNEAT*. This will also provide a direction for the final phase of the project.

The entire summer term should be dedicated to solving this problem using any new or already discovered approaches. Since the predator-vs-prey problem is known to be challenging even with neuroevolution technologies, this should expectedly occupy the majority of the summer term. Any extra time that is left of will be used to further perfect the algorithm, or be used to find a new but more difficult experiment.

2.1.5 Further development

If time is allowed, a novel task should be chosen to test the limit of *ExNEAT* in more computationally demanding tasks that involve a more complex environment. This new task should be largely different from the maze exploration and predator-vs-prey problem in the way that simulation should not be confine to a 2D, discrete-time space. A 3 dimensional environment is preferred as it better mimics a real world situation. Some good example could include dynamic control tasks in continuous time space such as double pole balancing and legged robots.

2.1.6 Summary

The final outcome of the project is summarized with the following expectations:

- Develop a complete pipeline for the training and testing of *ExNEAT* agents in any custom environment.
- Demonstrate *ExNEAT* excels at basic navigation tasks.
- Demonstrate the advantages of *ExNEAT* over NEAT in more difficult tasks that involve both navigation and other actions.
- Assess the generalization of *ExNEAT* in a much more demanding task and use this to further demonstrate the strength of *ExNEAT* while also highlight any possible drawbacks of *ExNEAT*.
- Characterize the evolution behavior of NEAT and *ExNEAT*, and perform analysis using the experimental results on why *ExNEAT* offers this level of performance.

3

ExNEAT

Contents

3.1 Analysis and design

3.1.1 Overview

The main limitations of ModularNEAT are that the range of tasks it can perform is confined to a single-objective, non-sequential task. Moreover, despite the fact that the paper has mentioned that ModularNEAT has explored spatial advantage such as the rotational symmetry of a 2D grid, the all-to-all, input-to-output connection makes it difficult for the network to explore true spatial dependency, such as telling from left to right. This is a key problem in neural networks since the input orders does not matter in a single network hence geometric information is lost during the training of a large neural network. This means ModularNEAT is likely to be only suited for environment that is spatially invariant and highly regular. Such as board games or image processing.

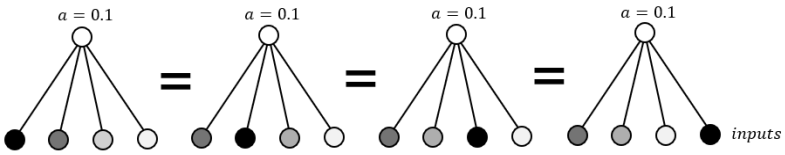


Figure 3.1: Illustration of the spatial independence of dense neural networks. Training 4 networks on 4 sets of inputs with the same value but different orders leads to all networks producing identical output.

To address the above issue, we purpose the *Ex-ModularNEAT* or *ExNEAT* in short. It is an algorithm developed using the original NEAT algorithm as foundation. As explained in section 1.2.3 that a NEAT algorithm tried to optimize a single network by evolving all its weights and connectivity. This algorithm differs from NEAT in the way that it instead evolves a number of smaller networks called *modules* (see Figure 3.2). Each module acts as a "sensor" that is responsible for processing inputs from a specific location. Depending on the complexity of the agent, the total number of modules can increase according to the nature of the inputs. For instance, for a humanoid robot, there may be 1 module for each leg and arm, 1 module for the body and 1 module for the head. During the evolution process, the modules are co-evolved together. This means that they will synergize together to achieve a common goal.

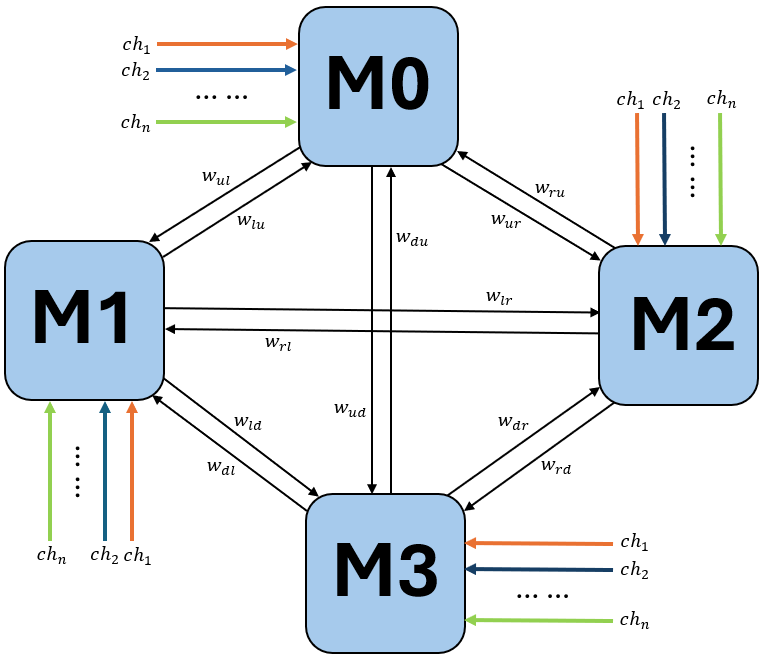


Figure 3.2: Overview of an *ExNEAT* agent. Here we displayed a 4 module agent but in practice it can have an arbitrary number of modules. The agent has sensors from different directions, for instance, left, right, up and down. Each module receives a multimodal input (indicated by the ch_1 to ch_n). Each module M_n is a separate neural network evolved through NEAT. The black arrows corresponds to a potential interconnection between the modules, which are jointly evolved along with the individual block.

An upgraded version which has an additional central module is used in the Ant experiment (see section 3.2.3 for more details).

The main idea is to explore the geometric implication of the sensor locations and its influence on exploration and agent learning, with the particular focus on the evolution process of the network within each module. The variation in sensor geometry will open up a more diverse genome selection and thus will encourage the discovery of more robust agents. Overall, those changes meant that Ant agents will utilize the full potential of an *ExNEAT* evolution system. This would allow the full potential of *ExNEAT* to be demonstrated and any outstanding strengths or drawbacks can be identified through the experiments. After initializing the population: $P_{init} = [p_0, p_1, p_2, p_3]$, a geometry vector v is used to create a symmetrical pattern between the modules. For a 4 module case, the geometries can be defined as:

- **Single**: where all the blocks are identical and: $v = [0, 0, 0, 0]$
- **FB** where the 2 front and back leg modules are identical and: $v = [0, 0, 1, 1]$
- **LR** where the 2 left and right leg modules are identical and: $v = [0, 1, 0, 1]$
- **Cross** where the diagonal leg modules are identical and: $v = [0, 1, 1, 1]$
- **ALL** where all 4 modules are different and: $v = [0, 1, 2, 3]$

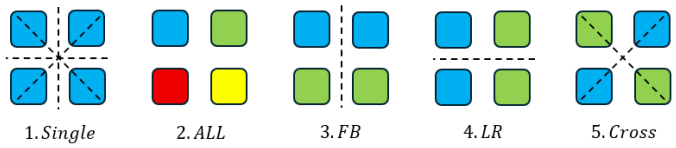


Figure 3.3: Illustration of the different geometrical configuration.

The chosen populations is then: $p_0, p_1, p_2, p_3 = p[v[0]], p[v[1]], p[n[2]], p[v[3]]$. The main body population will remain unchanged.

Among the configurations, the "ALL" configuration is to most versatile as it allows the agent to form any geometrical pattern. Because all 4 modules are initialized differently, they have the opportunity to automatically evolve into the best fitting pattern. Its effect is similar to the concept of CPPN in HyperNEAT [42] where a geometric pattern is evolved across the networks. However, unlink HyperNEAT, *ExNEAT* does not need an extra population of pattern producing networks and the geometric pattern acts on the modules instead on the neurons. The "ALL" configuration

should be most suitable in environments where the optimal geometry is difficult to define manually. While the other 4 configurations can be used when there is an obvious choice of module pattern.

Moreover, this method explores the inter-dependency between the modules by allowing them to interact with other modules without disrupting its own task. There are 2 types of inter-module connection: hidden neuron-to-input neuron (semi-recurrent) and output neuron-to-input neuron (recurrent). In this way, we can optimize the entire network by deciding whether we want the modules to function independently or with interconnections.

3.1.2 Algorithm implementation

The algorithm of *ExNEAT* is summarized below:

1. Based on the number of sensing locations, initialize a group of N populations of sensing modules, P_{init}. Each population contains I individual genomes. These population forms a *population group* from which genomes can be sampled to form different geometrical patterns. N depends on the minimum modules needed to complete each task. Within each population P_i, genomes that have the same index form a *genome group* G^i. Each sensing module is a feed-forward or recurrent neural network evolved using NEAT.

2. Configure the module geometry by forming a new population group of size N from the initial population group. Choosing identical populations for the sensor modules encourages a pattern while choosing different populations for each modules allows module diversity.

3. Initialize the connectivity between the modules. The connections could be from output to input (recurrent), from hidden to input (embedded) or unconnected.

4. Explore the environment and compute the fitness for each genome group to produce child networks genomes G_{nxt} from the previous genomes G_{old}. Each population are evolved independently and the evolved population forms the new population group P_{nxt}.

5. Repeat step 3 until a desirable fitness score is reached, or else the process is reset. For each repeat, if a genome group managed to achieve a highest fitness than the current best fitness, that genome group is saved. The best genome group g_{best} is also saved at the end.

Since NEAT only supports single agent evolution, a coevolution of 4 modules requires designing a custom evolution run function. Figure 3.3 summarizes this process.

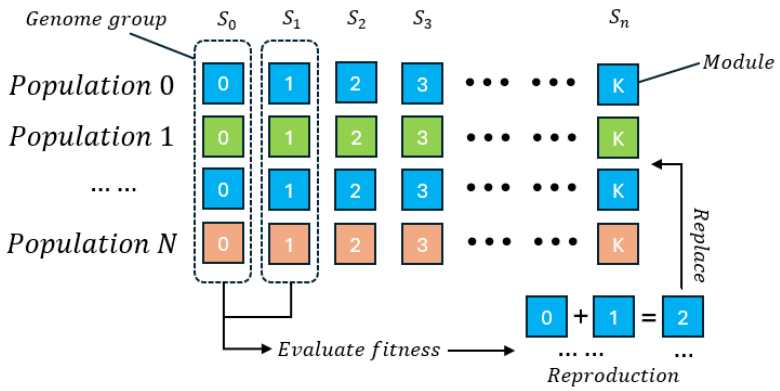


Figure 3.4: The Evolution process of *ExNEAT*.

At the start, we initialize a group of N populations where N is determined by the number of possible modules needed (Figure 3.4). Each individual genomes are then sampled from the population groups to form genome groups. The population within that group can be identical (same colour) or different (different colour). The option to set some population to be identical enables custom geometrical configuration. The selection and reproduction process is analogous to the NEAT algorithm, but instead a group of N genomes are evaluated jointly. Each genome within the same group receives a mutual fitness. And the genome groups with the highest fitness are selected for crossover and mutation, as in a typical genetic algorithm. The groups with the lowest fitness will later be replaced by the child genomes. For all the experiments in this report, the agents will require at most 4 modules for each sensing direction and an additional central module which may be used for a more stable control (see Figure 3.2).

One important step in the algorithm is to separate the inputs X based on their spatial properties. This is so that each modules will receive the parts of the input that corresponds to its sensing target. For navigation tasks, the input can be divided into the directions that the agents can travel, such as left and right. For control related tasks, the inputs can be distributed according to which part of the robot that they are responsible for. For instance, a leg robot may have sensing inputs from its 4 legs and the main body. Hence, the inputs should be regrouped as 4 sets of leg inputs and 1 set of main body inputs. In this way, each module at their respective location will receive the inputs that matches their own geometry.

Theoretically, this new approach would offer the following improvements on top of the original NEAT algorithm:

1. Because module patterns are simpler than neuron patterns, *ExNEAT* can drastically reduce the search space and would lead to an increase in training speed.

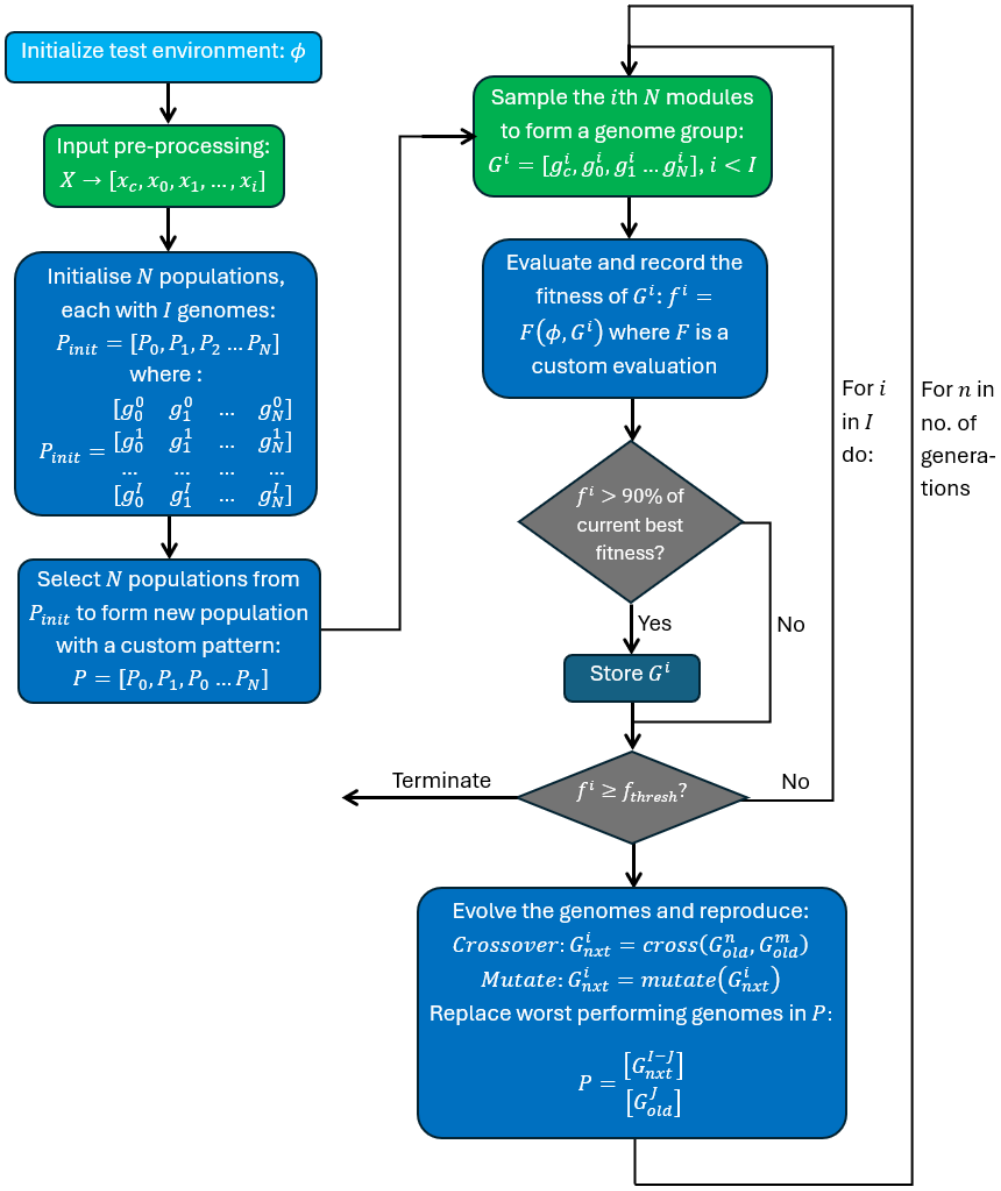


Figure 3.5: Overview of the algorithm implementation of *ExNEAT*. The main evolution function such as *crossover* and *mutate* came from the NEAT-python packages. But the function to run evolution was custom built to suit the new approach. Similar to a *NEAT* algorithm, the population P is evolved for a finite no. of generations.

2. Depending on the sensor positions, a sensor network called *module* is initialized at each sensing location. In this way, the entire architecture is subdivided into smaller sections of individual networks. This enables each module to be responsible only for their local detection and functionality, which makes the final decision making process more accurate and simplified.

3. Because each of the modules is now responsible for a smaller amount of tasks, their architecture can be simpler and overall the model size is reduced by orders of magnitude. Ultimately,

this leads to reduced overhead and a better memory efficiency.

4. The error rate would be drastically reduced. This is because if one module makes an incorrect prediction, the other modules are still able to average out the erroneous information and still make the correct decision overall. However, this is not the case for NEAT or ModularNEAT, where neurons or modules are closed dependent to each other. In effect, the agent making less mistakes ultimately means that the convergence rate is much faster.

Furthermore, each module is now only required to deal with a smaller amount of input data, the size of the overall network can be reduced significantly when compared to NEAT, while giving a faster convergence time. Moreover, since the exploration follows a pattern defined by a Cartesian space, the use of symmetrical modules within the network results in a more accurate decision making process as it maintains the same level of attention to every sensing directions.

3.2 Experiment design and implementation

3.2.1 Maze solving experiment

Introduction

As mentioned previously, *ExNEAT* is especially useful when it comes to navigation using multi-modal sensing process. The agents may receive signals of various types and each signal type has its own unique meaning. This would require the model to have different groups of sensors located at each critical position of sensing. To best mimic this effect while assigning a meaningful task to our agent, we used multimodal mazes as our experiment test bench.

The multimodal mazes as its name suggests, is a simulation environment in which the "floor" of the maze (called a *tile*) emits a two-channel signal which acts as "hints" for the agent to decide whether or not it should advance forward to a certain area. In our experiment, the mazes were designed so that they consist of square blocks in a Cartesian space. The agent always seeks the area with the highest signal magnitude. The goal is to reach the block where the signal strength is the strongest. The agent has a sensor on each side. Each sensor has 2 channels (ch. 1 and ch. 2) which senses the 2-dimensional signal from the tiles. During the experiment, the agent senses the signals emitted from the tiles on each side. All the tiles in the maze emit the same signal from channel 1. However, the tiles that are at the goal side of the maze also emit signals from channel 2 (Figure 3.6). This means the agent will not be able to infer the correct direction using only 1 channel since their signals are indistinguishable. It's only by utilizing signals from both channels will the agent be able to decide the correct direction to travel. This forces the agent to evolve to process multi-modal information. The *exploration steps*, t_{exp} is an important parameter since it affects the probability of the agent's completion of the tasks. For our experiments, this parameter value is determined by the minimum number of steps to solve the maze plus 2. Below is a summary of the 2 types of mazes used:

1. 1D maze: where the agent has only the sensing information coming from the left and right. Additionally, the agent can only move left or right. The agent always starts from the middle and one block is randomly selected to be the target block.

2. 2D maze: where the agent has sensing information from left, right, up and down. Each direction has 2 channels of signals. The maze would randomly assign one corner to possess the

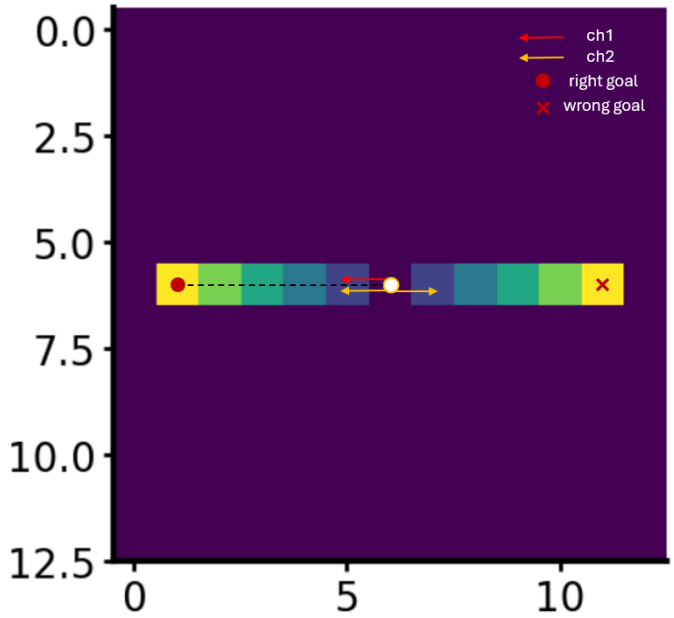


Figure 3.6: Illustration of the 1D maze of length 13. The agent starts at the center, which has the lowest signal ch. 1 intensity. The ch. 1 signal intensity is indicated by the color of the tile, where yellow is the highest. The agent's aim is to move to the left most side of the track (indicated by the red dot). The right hand side also has the highest intensity in ch. 1 signal strength, but it's an incorrect direction and the only way for the agent to distinguish between the two is to use ch. 2. The RHS of the track has only 1 channel activated and the other channel is pure noise while the other side has useful signals coming from both channels. Hence this environment will require the agent to evolve by learning to make decisions based on multimodal inputs.

highest magnitude signal block and the agent would randomly start at one of the remaining 3 corners. However, the agent does not have the information of its exact location during the exploration. Hence, the starting and ending position of the agent in the maze does not matter.

Simulation environment setup

Similar to the Frozen Lake gym simulation, the test environment is defined as a map indexed by an $N \times N$ grid where 1 indicating an empty space (called path) that the agent can travel to and 0 being a wall which the agent cannot occupy. (see Figure 3.3) For each coordinate on the grid (referred to as a tile), we assign it a 3 dimensional vector $v_i = [s_i, n_i, c_i]$ where $s_i \in \Re$ is the useful signal channel, $n_i \in \Re$ is a channel that emits either signal or noise, and $c_i \in [0, 1]$ is used to define a path or a wall. The maps are randomly initialized for each trial. The 1D maze is generated as simple, single track with walls surrounding it; The 2D maze is initialized using the Aldous-Broder algorithm [44] to prevent duplicity in the test environment. For each time step, the agent senses all its first neighboring tiles, namely the

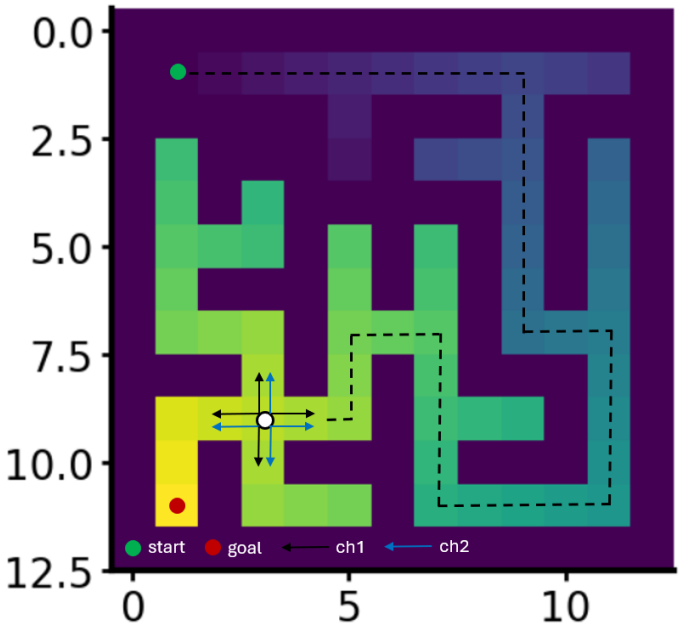


Figure 3.7: Example of a 2D maze of size 13x13. The goal position has the highest signal strength in both channel 1 and 2. The other sub-optimal goal positions have relatively lower signal intensity in channel 1 or 2. Like the 1D maze task, the agent will have to use signal from both channels to predict which direction will lead to the area with highest signal strength.

first tile that is above, below, to the left and right of the agent's current location. The vector v_i serves as the input for the agent to predict its next movement. The movement vector represented by $a_i = [U, D, L, R, S]$. Each letters corresponds to move up, down, left, right or stay at the same position. At each time step, the agent chooses one of the actions and moves one tile only. The aim would be to seek out the tile with the highest useful signal intensity, or $max(s_i)$.

The fitness of an agent at the end of a trial is calculated using the ratio of the difference between the actual time taken and the fastest possible time, and the maximum time allowed for exploration:

$$f_i = \frac{t_{actual} - t_{fastest}}{t_{allowed}}$$

Agent training

The *ExNEAT* agents used solve the *maze navigation* task and the *predator-vs-prey experiment* in the next section are configured to have an "Single" geometry (see Figure 3.3) with no interconnections. This means to assign identical genomes for each modules and leave the interconnection weights w_{ij} uninitialized. (see Figure 3.2). This design choice was due to

the simplicity of the nature of the experiments. Over-complicating the evolution process too much may lead to longer training time with little to no improvements, or potentially worse solutions than if all modules are kept the same. The "Single" configuration is sufficient to solve a 1D or 2D exploration task that has only 2 channels per sensor because the optimal configuration is easy to identify. Moreover, the purpose of the first 2 experiments were used to show that *ExNEAT* can significantly improve the performance of NEAT agents, regardless of its complexity.

3.2.2 Predator-vs-prey experiment

Introduction

In the second phase of the research project, we will deploy our model in more complex environments to solve more demanding tasks. The first task we have chosen is the predator-vs-prey experiment. In this environment, the agent will learn to solve different sub-tasks - navigating while foraging and hunting. In the foraging task, there are randomly initialized food sources in the map. The food sources are fixed and emit a multi-model signal that can be sensed by the agents. The goal of the agent is to collect all food sources within a limited amount of time. In the hunting tasks, the food sources will be moving in the map with either Brownian or Levy motion. The behavior of the preys can be configured using the experiment parameters, which were explained below:

(a) *noise*: This refers to a continuous sensor noise that has been applied to the input of the agents.

(b) p_e: This is the probability of emission. It controls how frequent the preys would produce the sensory cues.

(c) p_c: This is the probability of decay. It governs how fast the prey's emission will be disappearing after emission.

(d) p_m: (Only applies to hunting tasks). This is the probability of motion and it controls how rapid the preys will be moving.

Unlike the maze solving task, where there exists only a single goal position, the pvp environment requires the agent to repeatedly sense and relocate to track the path taken by the preys. It therefore requires the agent to evolve a more generalizable network that can control the agent to travel with higher degree of freedom and precision.

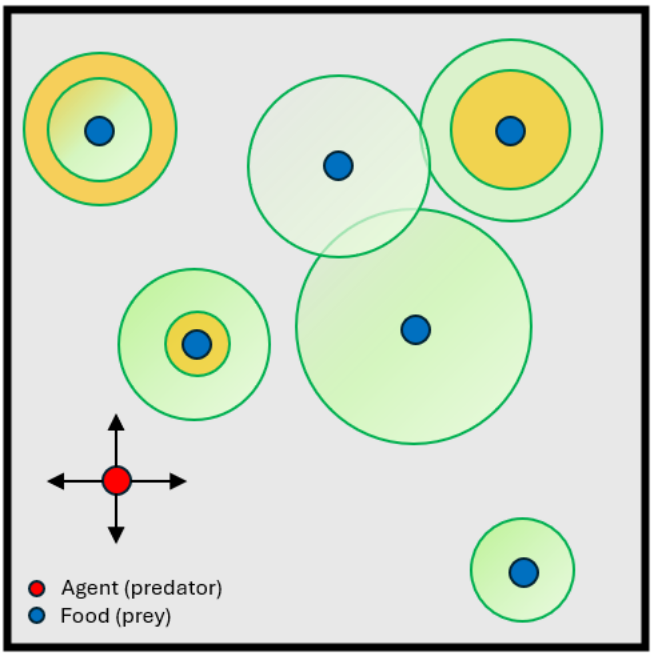


Figure 3.8: An illustration of the pvp task. The prey (represented by the blue dots) emits two types of signals: odor (orange circle) and sound (green circle). These two types of signals act as sensory cues that will attract the agent to capture the prey. The odor and sound signals are emitted as a Gaussian function, meaning at any given time the magnitude of the odor and sound signals are random and decays from the radius of the prey. This requires the agent to compare the amplitude of both signals and makes prediction about how likely the prey may be present at a particular location, and where exactly is the prey located.

Experiment setup

The map of the pvp experiment is defined in a similar fashion as the maze navigation task. But instead, walls only appear around the edge of the map to confine the agent and the preys. Apart from that, there are prey objects being added to the map. Each prey emits sound s_i and odor o_i. Both are used as input signals for the agent to locate and track the preys. The prey can be configured to be stationary or mobile in the environment. At each time step, both the agent and the preys are only allowed to advance across one tile.

If, at any time step, the agent arrives at the same coordinate as the prey, the prey is marked as caught. A list is used to store the state of all the preys with 1 being free and 0 being caught, the fitness is the ratio of caught preys at the end of the trial.

$$f_i = \frac{len(caught\_preys)}{len(all\_preys)}$$

3.2.3 The Python Gymnasium-Mujoco environment

Overview

OpenAI's gym (now called gymnasium) package is an open sourced, Python-based simulation environment with easy installation and quick deployment. It provides a library of famous reinforcement learning benchmark tasks such as the double inverted pendulum mountain bike and bipedal walker experiment. What makes it more suitable is that all the environments are programmed to exhibit physically realistic simulation while only using CPUs backend.

The Ant experiment is a 3-dimensional task within the Mujoco library. In which a quadruped robot is controlled to move in a flat plane. Each leg of the robot consists of 2 linked and 2 joints while the robot's main body is a single rigid object. The aim of the Ant experiment is to maximize distance traveled by the agent to the x-direction (to the right) in a fixed amount of time.

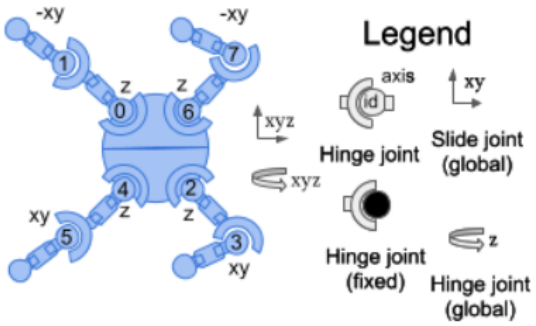


Figure 3.9: Ant agent physical parts summary.

At every simulation time step, the agent senses its current state in the form of a 105 dimensional vector named observation space, *obs*. The observation is subdivided into different sections that correspond to the position, velocity, force and torque of the main body and the 4 legs: $obs = [obs_{main}, obs_{leg0}, obs_{leg1}, obs_{leg2}, obs_{leg3}]$. The action space of the agent is an 8 dimensional vector, a that corresponds to the torque output of each hinge joint, T_i. Where $-1 \leq T_i \leq 1$.

The Ant experiment preserves the goal of multi-modal learning since its input is a mixture of physical quantities such as position, velocity, force and torque. It also has a regular geometrical shape that can demonstrate the advantage of *ExNEAT*. Thus, this was chosen to be the final challenge of *ExNEAT* in evolving multi-model systems. A significant difference that Ant gym makes is that in PVP and maze solving experiments, the system is non-dynamic

(there is no control-related tasks), low in resolution (can be solved in less than 100 time steps) and of lower dimensionality (input size at most 2D); whereas in the Ant experiment the agent will control the ant's stability while moving, the input dimensions can reach as high as 34D and simulation lasts around 500k time steps. Hence, this design decision could lead to potentially two outcomes, either: (1) *ExNEAT* can still achieve decent performance as in the previous 2 experiments and is still able to outperform NEAT. In this case, clear comparison should be drawn to highlight the strength of *ExNEAT* and to explain why it performs better. Moreover, any possible adjustments should be highlighted to explore any potential weaknesses and further improvements. (2) *ExNEAT* agent performs poorly and its performance is worse than NEAT agents. In which case, detailed explanation should be given to show why *ExNEAT* might not perform as well as in the previous experiments. A comparison will still be required to exhibit any evolution difference between *ExNEAT* and NEAT, and will allow for further improvements in the future. But regardless of the outcome, the Ant experiment would be an excellent test environment for any deep learning agents that aim to learn a control and navigation task. And any results obtained will be valuable in terms of researching evolutionary algorithms in robotics.

Experiment setup

The main goal of the agent is to advance as far as possible in the given time, however, the Mujoco environment also takes into account the other aspects to measure the agent's performance such as the magnitude of the force applied by the agent and the extend of movement. The agent will be rewarded for traveling longer distances and penalized for making angular movement or forces that are too large. The influence of the rewards can be controlled by the internal weights of the Mujoco agents, hence the reward function for ANT environment is:

$$R = w_{forward}R_{forward} + w_m R_m + w_{ctrl}R_{ctrl} + R_{healthy}$$

Where the last term $R_{healthy}$ measures whether the agent survives during training and must be kept high at all times. In our training experiments, we only use the forward fitness $R_{forward}$ when assigning fitness to each genome. This is to prevent the agent from overfitting through devoting too much attention to the last 2 rewards while ignoring the fact that moving forward is the most important task. For the testing of the trained agents, we used the full fitness R and set the weights according to their importance to the end goal, namely:

$w_{forward} = 1$, $w_m = 0.01$ and $w_{ctrl} = 5 \times 10^{-4}$.

Early termination strategy

At the start of the training, a significant proportion of the genomes would produce agents that were unable move at all. It is therefore important that these defective agents are eliminated early on during the training process in order to save computation time and avoid contaminating the genome pool. The reason behind this is that the Ant agents not only received reward for moving forward, but also for making a more gentle movement. Hence, some agents may have a relatively high fitness but still perform inadequately at walking. There are effective ways to prevent this from happening. The Mujoco Ant environment does have its own termination strategy, but it only terminates the simulation if the ant becomes unhealthy. But in practice, an agent can be both healthy and immobile at the same time. Therefore, we introduced an early stopping strategy to filter out the genomes that may have caused the ant to become immobile. The strategy is that, instead of using the full reward, we only use the forward reward to measure the capability of the agent. This termination rule ensures that if an agent has not made any forward movement for 100 time steps, the training is terminated. This saves the time of having to train for 5000 time steps. And finally, we can use the agents trained using only forward reward to test its generalization in an environment where the fitness is calculated using the full reward.

Agent training implementation

In the Ant experiment, the *ExNEAT* system is upgraded to have an additional central module. The idea is that the 4 leg modules act as sensors while the central module makes the final decision. This will allow for more stable control of the agent as this new experiment involves the precise control of the robot's dynamic states. Moreover, unlike the agents in maze or predator-vs-prey experiment, the Ant robot is a multi-body agent. Therefore, it makes sense to divide the sensor information into 5 sections: 4 leg sensors and 1 main body sensor. This means that up to 5 different genomes pools will be used to train the Ant agent. And most importantly, the Ant agents will use the "ALL" configuration to explore better network topologies. This is because, unlike the maze and predator-vs-prey experiments, it is difficult to identify the optimal geometry of the Ant agnet's networks. And by comparing the performance of agents trained with "Single", "ALL" and other configurations will allow the strengths of custom-evolved geometry to be demonstrated.

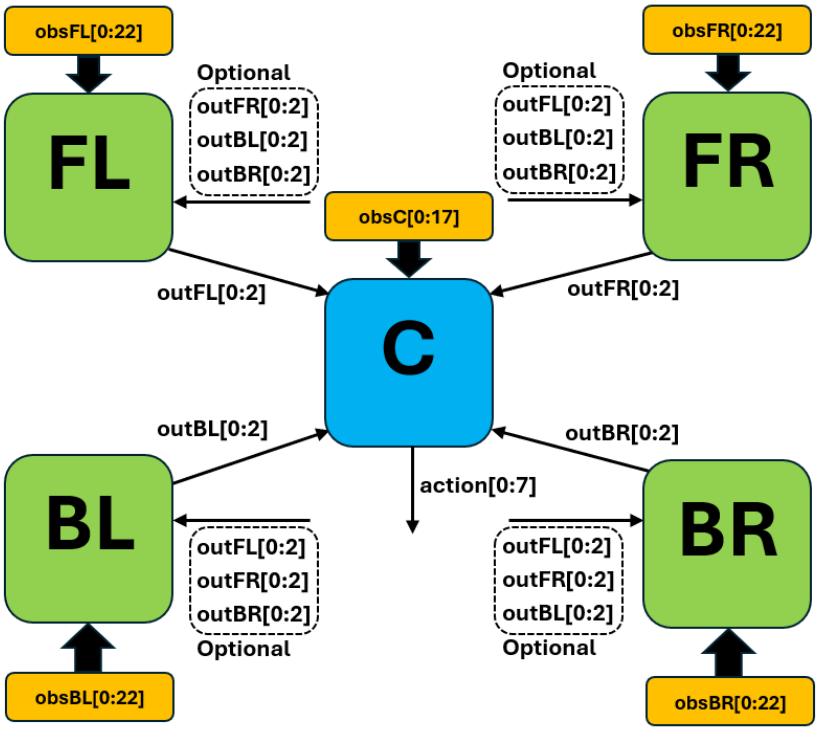


Figure 3.10: Ant module-network connection map. We divided the module locations according to each leg's position. As can be seen from the figure, there can be 4 leg modules and a central module block. Each leg module receives their respective obs_{leg} as input while the central block receives obs_{main} as input as well as the 4 outputs from each leg modules. Each leg modules also have an optional recurrent connection from all the other leg modules. The central block outputs the final action space.

Apart from the fifth module configuration, the *ExNEAT* system was upgraded to have the following flexibilities:

- The Ant agents are allowed to have any geometric configuration (see Figure 3.8), each leg modules can be different to each other, symmetrical, or identical.
- Added the option to switch the maximum number of modules between 4 and 5.
- Added the option to interconnect or unconnect the modules.
- Added the option to change the output data width.
- Added the option to form a recurrent connection from the central module to the leg modules.

The training process of the 5 module *ExNEAT* is similar to the 4 module case (Figure 3.5):

(a) Initializes 5 populations of genomes: $P_{init} = [p_0, p_1, p_2, p_3, p_C]$.

(b) Choose a geometry vector v to create pattern between the modules.

(c) Form new population: $p_0, p_1, p_2, p_3 = p[v[0]], p[v[1]], p[n[2]], p[v[3]]$. The main body population remain unchanged.

(d) For every 4-genomes groups in (p_0, p_1, p_2, p_3), compute the fitness 10 times and take the average result.

(e) Evolve the populations using standard NEAT methods.

Also, unlike in the previous 2 experiments, in an Ant agent each network is a feedforward neural network instead of a recurrent network. This is due to the fact that the Ant agent does not need to memorizer previous information to make decisions. For example, in a maze experiment, there are period of time where the tile gives no information; in a predator vs prey experiment, the agent need to remember the trials emitted by the preys to infer their previous locations, etc. In the Ant experiment, the agent focus solely on moving forward while dealing with the robot's internal and external dynamics. Thus having recurrence will have little to no advantage. Each network has 4 hidden nodes which offers a good trade-off between generalization and network size.

4

Experiment results

Contents

4.1 Overview of training verification

In all three experiments, we will compare the training performance of ExNEAT with that of NEAT. Since there are different agent types for different experiments, we will train a set number of agents for each. For the maze and predator-vs-prey experiments, all agents used the ”Single” geometry configuration (Figure 3.3) and are classified using their training conditions. For instance, agents trained with different sensor noise levels, maze gap lengths or maze sizes. For the Ant agents,

the agents use any custom geometry configuration and hence are classified using their geometrical configurations. For instance, whether it is 4 modules or 5 modules, "ALL" or "Cross" configuration, modules interconnected or unconnected, etc.

General training procedure:

1. Select the agent type (in our case it's either *NEAT* or *ExNEAT*).
2. Select the training condition. E.g. high noise or bigger environment.
3. For the selected agents, generate a set number of environments. For the maze and predator-vs-prey experiments, the agents were trained for 10-20 trials. Each trial was initialized randomly and different to one another. For each repeat, the fitness score was computed and saved along with agent's genome and configuration.
4. repeat the previous step N times to obtain N sets of results. Ideally, N is between 5 and 10.
5. repeat step step 1-3 for different environment configurations such as different sensor noise levels or maze complexity.

For the Ant agents, there was an initial selection phase. During that phase all the classes of the Ant agents are trained for 1-2 times. This is because, except for the "ALL" configuration, the other geometric configurations may not be all suitable for the Ant task's agent and it is difficult to predict beforehand. Therefore, it was required to test all the geometrical configurations and select the ones that are able to solve the Ant task. We can then use those initial training performance to select the top performing agent classes and train them further to obtain an estimation of their average performances. This will allow us to understand why *ExNEAT* offers its level of performance compared to NEAT, and how does geometric structure affects a NE agent.

4.2 Overview of testing verification

The trained agents are tested for their performance under different types of disturbances in order to assess their robustness:

For the *maze experiment*, we measured the agent's ability to solve the maze when (1) The sensor data becomes increasingly noisy. This assesses the agent's robustness to sensor noise. (2) During certain time steps the sensors do not receive any data. This is to force the network to evolve

recurrent connections. (3) The complexity of the maze increases, for both NEAT and ExNEAT agents.

For the *predator-vs-prey task*, we assessed the agent's performance under the following criteria: (1) Robustness to noise. This is similar to the sensor noise robustness in the maze experiments. (2) Probability of movement, p_m: this assesses the agent's ability to capture more agile agents. (3) Probability of emission, p_e: p_e controlled the probability of a prey giving out a sensory cue. Varying this parameter assesses the agent's ability to locate the preys with minimal information. (4) Signal decay rate, p_c: this parameter governs how fast the sensory cues are diminishing. Changing this parameter will assess the agent's ability to memorize the trials emitted by the preys.

For the *Mujoco Ant experiment*, we will test the agent's robustness to noise as in the maze and predator-vs-prey experiments. Moreover, we will assess the agent's ability to (1) Move forward. This means to maximize the forward reward, $f_{forward}$. (2) Move forward and stably. This means to maximize the full reward, f_{full}. And since we have trained the Ant agents using the forward reward as fitness, it should also generalize well to achieve a decent full reward that takes control and forces into account.

All these tests will assess the agent's ability to solve the maze under non-ideal environments. They also ensure that the agents trained will have a certain level of robustness and and difference between NEAT and ExNEAT will be amplified. Thus allows for clearer comparisons later on.

Testing experiment:

1. Select an evaluation criterion (for instance, testing the agent's performance under noise, signal absence or simply how well it can generalize in larger environments).

2. Select a set of N genomes of the top performing agents from both *NEAT* and *ExNEAT* agents classes by first validating their performance using a test environment 10 times.

3. Assess the performance of the top agents trained using both algorithms by letting the agent solve 100 randomly generated environments with the specific criteria previously defined.

4. Create a plot showing the fitness score of both agent types to compare their performances.

4.3 1D track maze

4.3.1 Training performance

The 1D track maze task is a simple 1D navigation environment that was first used to verify the basic functionality of *ExNEAT*. Moreover, due to the symmetrical nature of *ExNEAT* agents, we would theoretically expect them to train faster and have a better performance than the original *NEAT* in the maze solving tasks. This first experiment can easily magnify the advantages of *ExNEAT* algorithm and can provide us with the next direction of the project. We trained agents that uses *ExNEAT* and *NEAT* respectively and recorded their training histories after 10 repeats. (Figure 2.1).

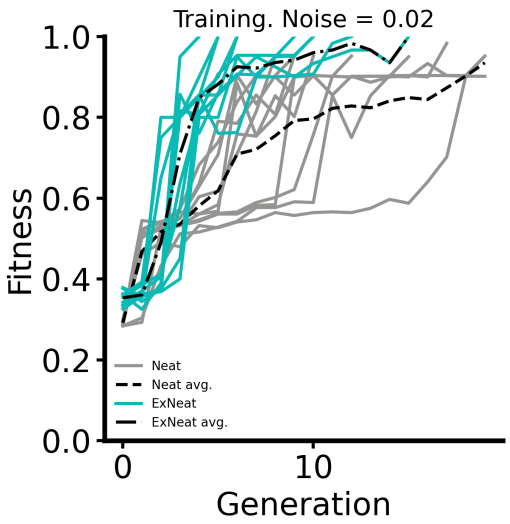
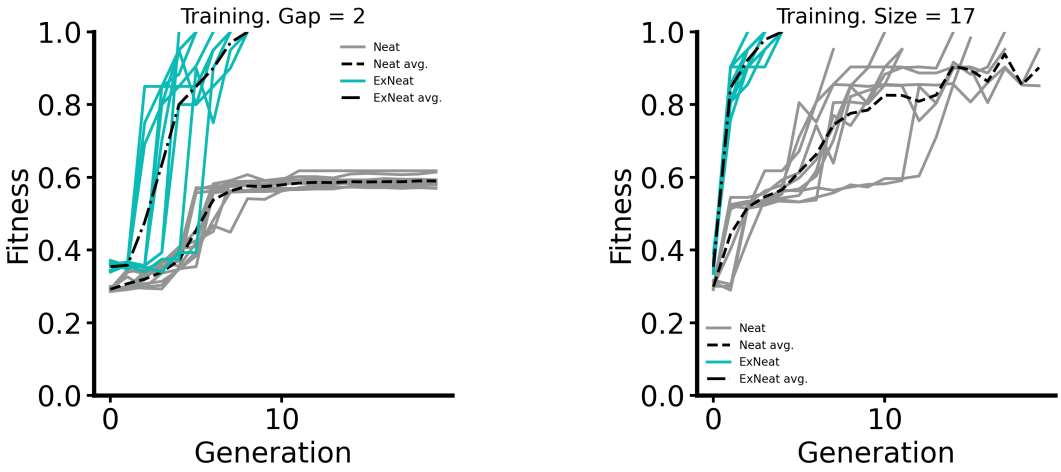


Figure 4.1: 1D maze. Trained with noisy sensors. It was found that the agent trained using *ExNEAT* reached the perfect fitness much faster. The overall fitness reached by the *ExNEAT* agents are also higher than NEAT.

The sensor noise is one key training parameter that influences the agent's behavior. However, to ensure that *ExNEAT* truly succeed NEAT in all aspects, we observed the training performance of both agents under the influence of other key training parameters.

It was found that by adding temporally missing sensor inputs (the maze gaps) or increasing the length of the mazes, the *ExNEAT* agents performs especially better then *NEAT* agent. (Figure 2.2 - 2.3).

We have demonstrated that in practice, *ExNEAT* achieves a much faster convergence when the



(a) Training under long duration of maze gap.

(b) Training under long maze length.

Figure 4.2: Training performance comparison for 1D maze.

training conditions are kept the same for both types of agents. As can be seen, NEAT is easily influenced by an increase in maze gap (Figure 4.2 a)) or an increase of maze length (Figure 4.2 b)) but ExNEAT maintains a good training performance.

4.3.2 Robustness test

From the training results, it seems certain that *ExNEAT* agents produce better agents than NEAT. But it is also possible that the solutions produced by both algorithms are of similar performance. Therefore, in order for *EXNEAT* to be proven as a better algorithm, it must also be able to maintain a higher performance in unseen environments. Hence, the generalization ability of the agents were assessed by computing the fitness score of the top performing agents while varying the environment training parameters. We tested the agent's performance by sweeping the sensor noise level, maze gap duration and maze length. These variables are important in determining the practicality of the agents especially when deployed in real world scenario where environment is usually unpredictable.

Noise robustness

We selected NEAT and ExNEAT models that were trained using sensor noise level of 0.05 with all the other hyperparameter kept the same. This ensures a certain level of resistance against noise for both agents and allows for a fair comparison. The black curve is a random agent and is used

to provide a baseline for the performance.

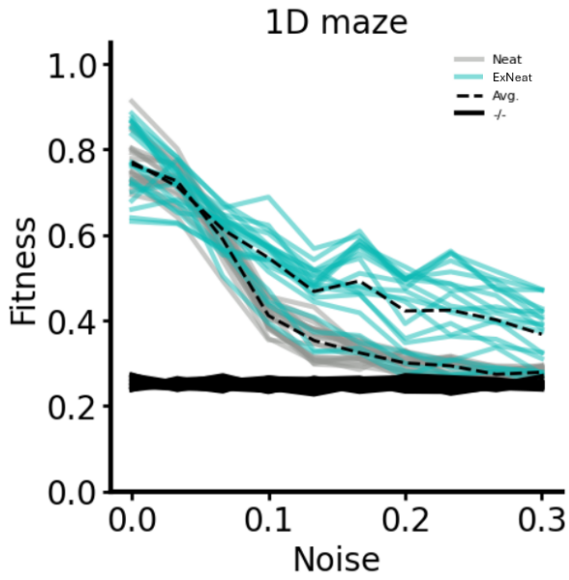


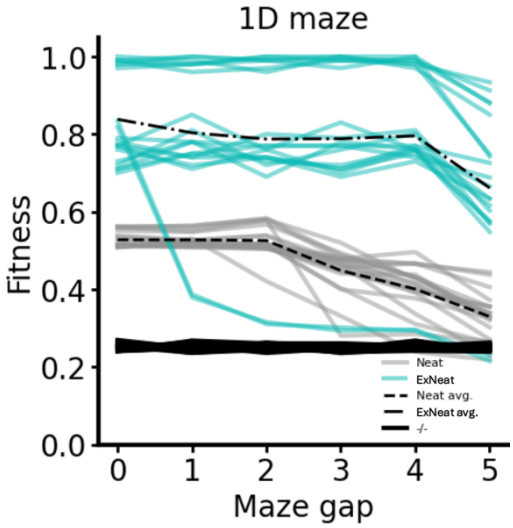
Figure 4.3: Comparison between ExNEAT and NEAT in solving the 1D maze task under different sensor noise level. *ExNEAT* agents have shown to be more robust to noise at higher noise scales.

Gap and maze length robustness

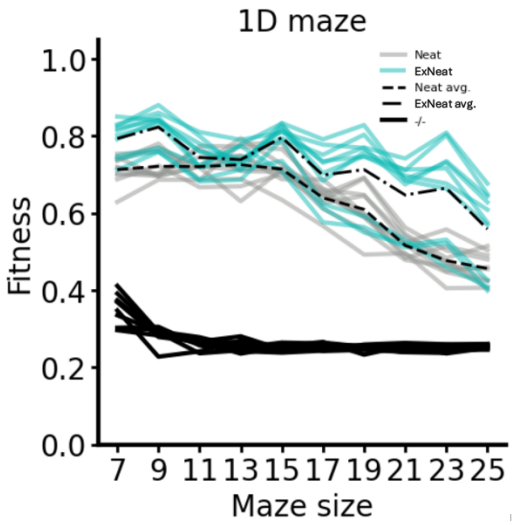
To further investigate the behavior of both agents under different scenarios, we made the tasks more complicated by increasing the maze gap and maze length respectively. This will test the agent's ability to: (1) utilize memory when there is a shortage of input data. (2) apply the same policy to longer mazes that were previously unseen during training.

ExNEAT agents achieved consistently a higher fitness score for all noise levels. It was also able to produce perfect fitness in a noise-free conditions while *NEAT* agents struggle to solve the maze properly. Furthermore, despite the increasing duration of data absence, *ExNEAT* agents managed to achieve a near perfect fitness for the majority of the trials, while *NEAT* agents were still unable to completely solve the maze under such conditions.

Overall, *ExNEAT* achieved consistently a higher training and testing performance under all environmental variations. In some cases, there was an almost two-fold improvement in training efficiency or testing fitness.



(a) Comparison between *ExNEAT* and NEAT in solving the 2D maze task under different sensor noise level. Both the NEAT and *ExNEAT* models were trained using maze gap 1 with all the other training hyperparameter kept the same.



(b) Comparison between *ExNEAT* and NEAT in solving the 1D maze task under different maze lengths. Both the NEAT and *ExNEAT* agents were trained using maze length 13 with all the other training parameters being kept the same.

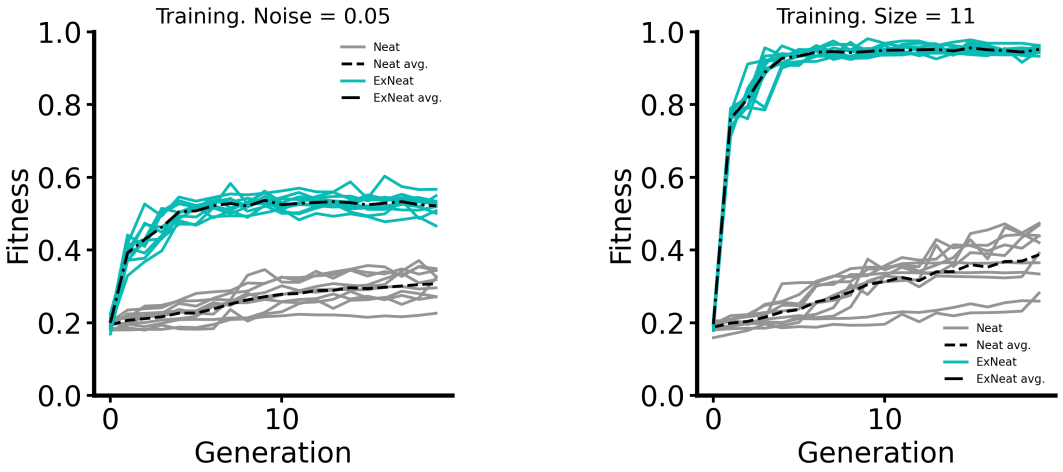
Figure 4.4: Training performance comparison for 1D maze.

4.4 2D general maze

The 1D maze was an over-simplified version of a maze solving task. However, in reality, most mazes are at least 2-dimensional. Therefore, to demonstrate that *ExNEAT* agents truly excel at solving navigation tasks, they should at least be able to solve the 2D maze task. The environment parameter setting is similar to that of 1D maze. The agents are tested by sweeping all the variables that can influence the performance of the agents, including noise level and maze size. For the 2D maze, there are no maze gaps because the goal of 2D maze focus more on navigation instead of decision making. The following results assess the 2 agent's performance in solving a 2D maze:

4.4.1 Training performance

ExNEAT still manages to out perform *NEAT* in both of the training conditions. In the second scenario where the maze complexity is excessively high, the *NEAT* agent is completely unable to solve the maze within a reasonable number of generations, however, the *ExNEAT* agent still achieves near perfect fitness regardless of the maze complexity.



(a) Training performance comparison in high noise environment.

(b) Training performance comparison with complex mazes.

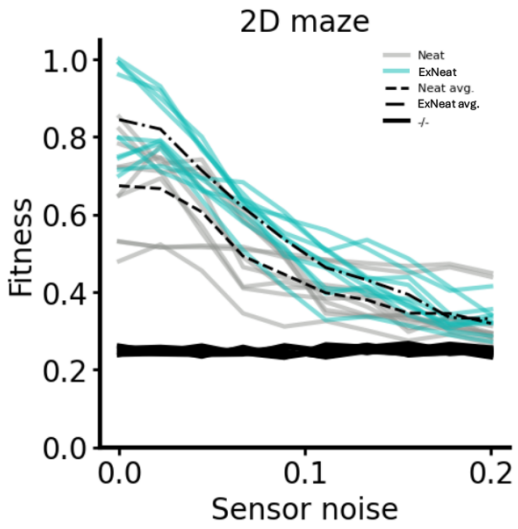
Figure 4.5: Training performance of agent in 2D maze environment. Similar to the 1D maze experiments, observed the agent's performance when being trained in a non-ideal environment such as high noise or long maze size.

4.4.2 Robustness test

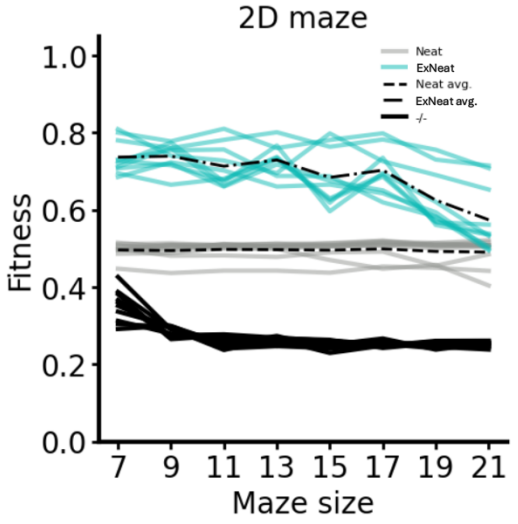
The previous agents were tested in environments where the conditions did not change. Still it is important to assess the agent's ability to adapt to changes. Hence, the next set of experiments will measure the agent's robustness against disturbances within the maze experiments to further verify the performance of *ExNEAT* algorithm. We tested the agent's behavior when the environment parameters differs from that of in training. Similar to the 1D maze, a range of sensor noise and maze size values were swept and the agent's fitness is plotted for each sweeping point.

Noise and size robustness

It can be seen from the figure that *ExNEAT* out-performed NEAT in both maze experiments. In some cases where all the top agents trained using NEAT were unable to solve the maze, the agents trained using *ExNEAT* were able to achieve perfect fitness in low disturbance conditions. Moreover, *ExNEAT* discovers the fittest solution using much less generations. Overall, it is also much more resistant to disturbances such as absence of signals or noise influence.



(a) Comparison between *ExNEAT* and NEAT in solving the 2D maze task under different sensor noise level. Both the NEAT and *ExNEAT* models were trained using sensor noise level of 0.05 with all the other hyperparameters kept the same.



(b) Comparison between *ExNEAT* and NEAT in solving the 2D maze task under different sensor noise level. Both the NEAT and *ExNEAT* models were trained with maze size 11x11 with all the other training parameters being kept the same.

Figure 4.6: Robustness test comparison of agents in 2D maze environment.

4.4.3 Conclusion

From the experimental results, it was evident that *ExNEAT* algorithms produced agents that not only require less generations to train, but also more robust against variations in the multi-modal maze navigation environment. In all cases, *ExNEAT* outperforms NEAT. Moreover, in many occasions, *ExNEAT* was able to offer a 2-3 fold improvement in final fitness.

4.5 Predator-and-prey experiment

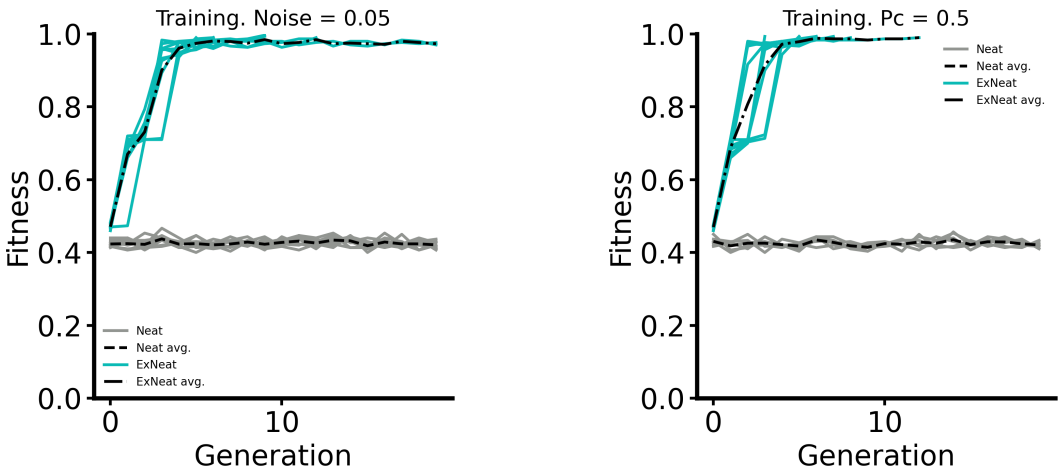
ExNEAT has demonstrated a great advantage in the maze navigation task. However, some of those experiments were simple enough to be solved by NEAT and other RL algorithms. It would be interesting to see if *ExNEAT* was able to excel at tasks that were difficult to solve for NEAT. Moreover, it would be useful to observe the behavior of *ExNEAT* agents in tasks that involve more than just navigation. Therefore, the predator-vs-prey was introduced (see section 3.2.2) to assess *ExNEAT* in more complex tasks. This experiment involves both locating the prey (navigation) and evolving a policy that maximized the number of preys captured (learning a decision making process).

4.5.1 Foraging

The first type of experiment is simplified and only involves static preys. Despite the simplicity of the experiment, NEAT agents were proved to be unable to solve it under 20 generations. This therefore would be served as another milestone for *ExNEAT* if it can solve a task that NEAT could not. And moreover, if it can still perform equally well as in the maze experiments. The agents were trained under environmental variations where a noise level of 0.02 is added to increase the difficulty of decision making. The second set involves training the agent to capture tricky preys. This was achieved by setting the decay rate of trials (p_c) from 0.0 to 0.5 (see section 3.2.2 for the parameter details). This way, the trials will exist for only a short amount of time. This allows us to assess the training performance of *ExNEAT* under the influence of environmental and prey conditions. And most importantly, to compare the performance between *ExNEAT* and NEAT in those adversarial environments.

Training

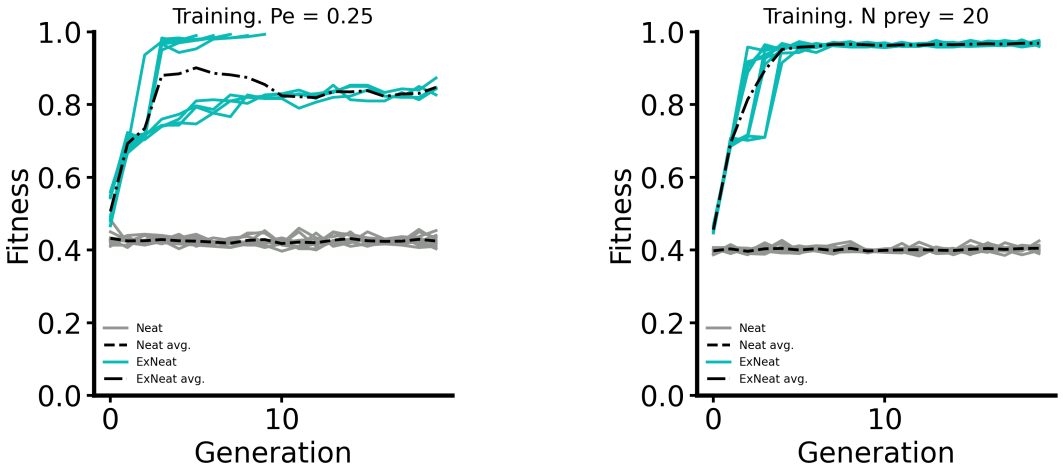
Both agents were trained for 20 generations. With the maximum steps allowed to explore the environment being 50. And it can be observed that, while ExNEAT reached a good fitness score within 10 generations, Neat agents were unable to progress in learning at all.



(a) Training performance with noisy sensors.

(b) Training performance with high pc.

Figure 4.7: Training performance of agent in foraging task environment (1).



(a) Training performance with more stealthy preys.

(b) Training performance with more preys.

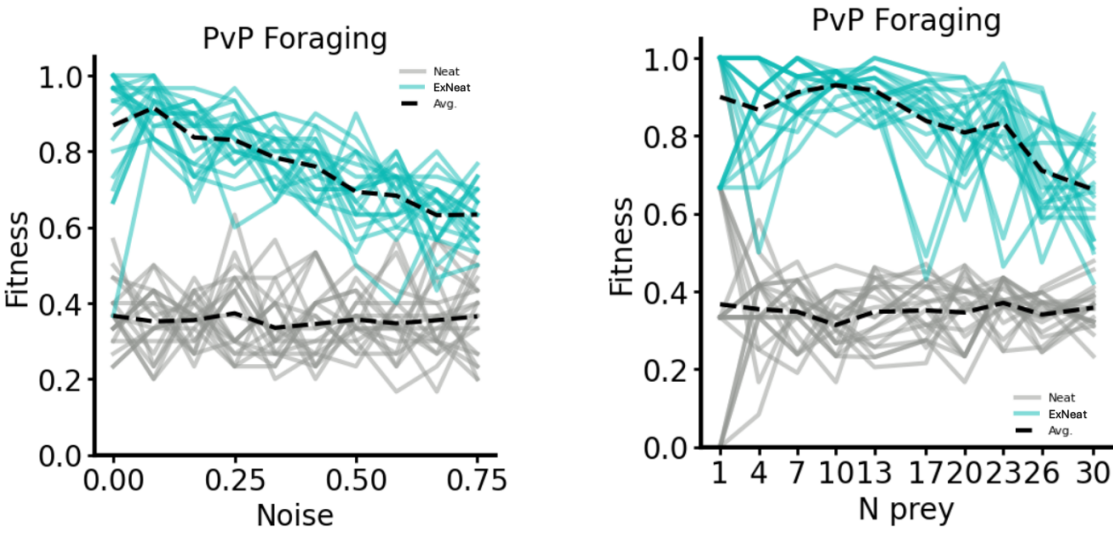
Figure 4.8: Training performance of agent in foraging task environment (2).

Robustness

To assess if the *ExNEAT* agents are better overall, their robustness were tested under environmental variations. We picked the agents that were trained with sensor noise 0.02, pc = 0.5, pe = 0.8 and pm = 0.5. This is so that we expect each agent type to have a near-perfect performance in ideal environments and an acceptable but degraded performance in the harsher environments. Like the maze experiments, these sets of tests focused on the environmental influence, namely: how good are the agents in resisting noise and what happens when there are more preys than it can catch in a given time?

The first 2 tests are performed by varying environmental influences. Although one may be interested to analyze the agent's behavior when the nature of the preys changes. So the last 2 sets focused on making the preys more difficult to be caught. This was done by sweeping the probability of emission (p_e) and rate of trial decay (p_c) and observing the fitness achieved in each case. These measurements are sufficient in determining the robustness of the agents when being deployed in unseen environments and therefore will justify that *ExNEAT* is a more superior method.

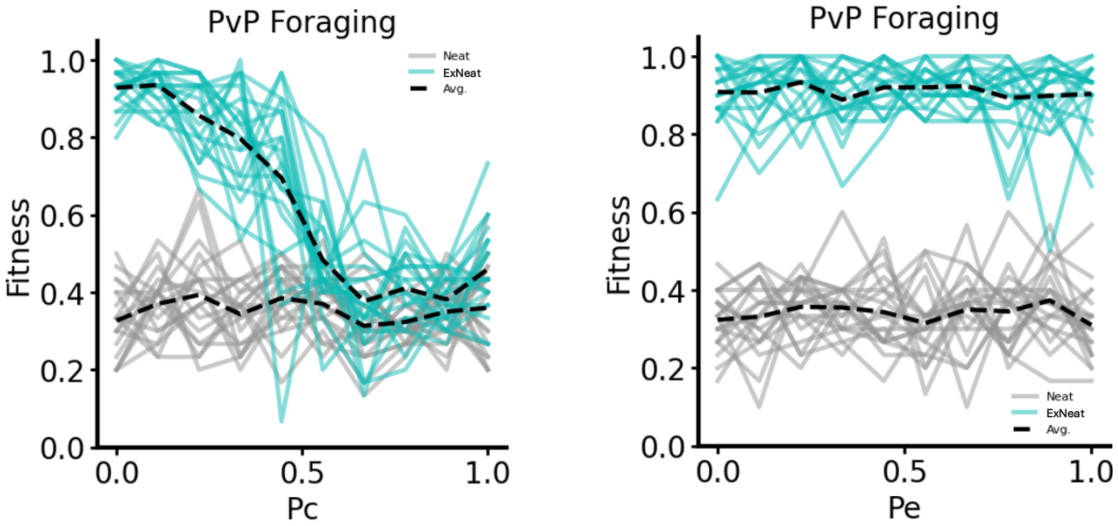
It is obvious from the result figures that *ExNEAT* agents are more robust in all cases. It is not surprising that NEAT agents were unable to solve the task at all due to their inadequate training performance early on. For the p_c sweeping test, the fitness of *ExNEAT* agents drops significantly at high values of p_c. This is expected as making p_c larger means the trials emitted by the preys will not be retained in the environment. With the absence of sensory cues, it is normal that the



(a) Robustness against sensor noise scale variation.

(b) Robustness against total prey number variation.

Figure 4.9: Testing performance comparison of agents in foraging task environment (1).



(a) Robustness against p_c variation.

(b) Robustness against p_e variation.

Figure 4.10: Testing performance comparison of agents in foraging task environment (2).

agents were unable to locate the preys.

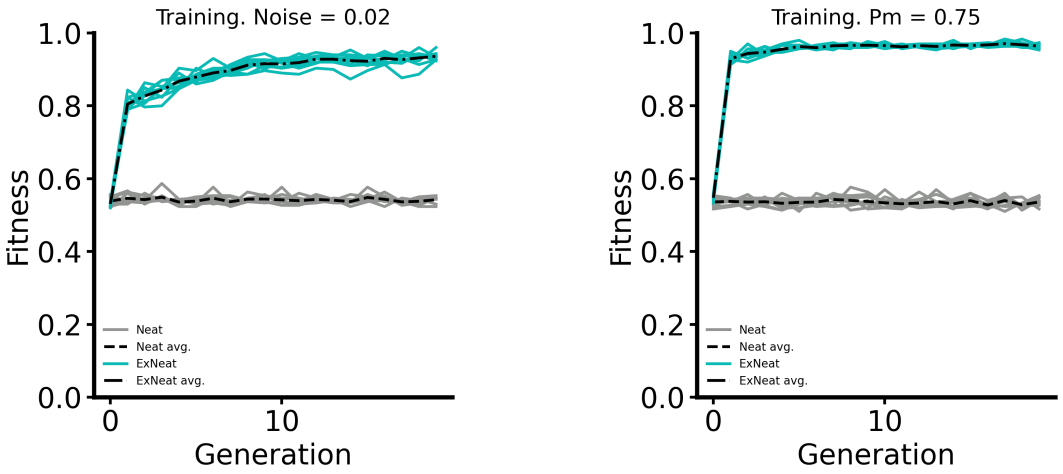
4.5.2 Hunting

The foraging tasks is a easier version of the predator-vs-prey experiment and was used to verify the *ExNEAT* agent's ability to multi-task. To fully verify the capability of *ExNEAT* algorithm, we then test them in the full predator-vs-prey task that involves moving preys. For the hunting tasks, there is an additional parameter that influences the agent's performance. This new parameter p_m

is the probability of movement of the preys. Setting a none zero Pm would allow the prey to change location at each times step and increases capturing difficulty. We expect the effect of increasing the probability of movement to be comparable to increasing the trial decay rate p_c. Both should worsen the overall performance of both agents. The food sources were moving with Levy motion. This is because it is a type of movement that more closely resembles the natural motion in animals.

Training

The training procedure follows the same fashion as the foraging experiments. Both agents were first trained in environment with high noise and rapid moving preys to verify their basic performance. Then they were trained in the other environmental uncertainties. The maximum steps allowed to explore the environment was set to 50.



(a) Training in high noise environment.

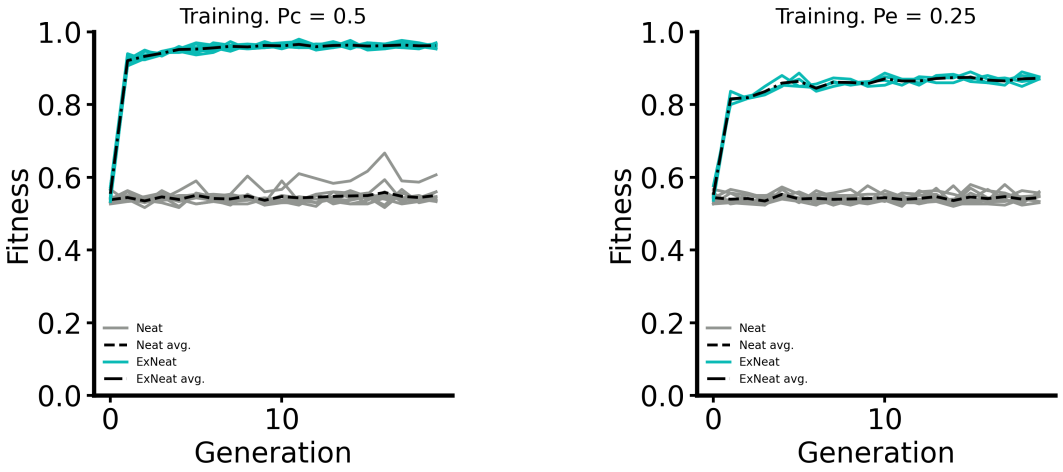
(b) Training with high mobility preys.

Figure 4.11: Training performance comparison of agents in hunting task (1).

ExNEAT was able to maintain the performance by converging to a near perfect fitness regardless of the training conditions. The NEAT agents on the other hand, were unable to solve the task at all.

Robustness

Like the foraging tasks, to fully verify the performance of *ExNEAT* agents, the agents' performance under noise, number of preys, Pe and Pc variations were assessed. Additionally, the probability of movement is swept to see if the effect that will have on each agent types; also the map size

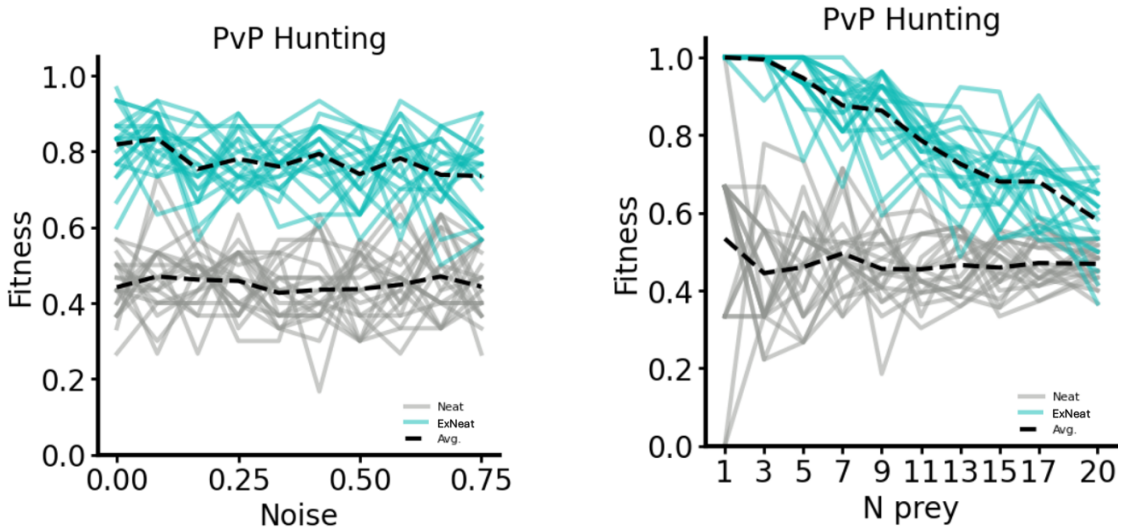


(a) Training with high prey signal decay rate.

(b) Training with low prey signal emission rate.

Figure 4.12: Training performance comparison of agents in hunting task (2).

was gradually increased so that the preys have more space to run. The performance of both agent types should in theory decrease gradually.

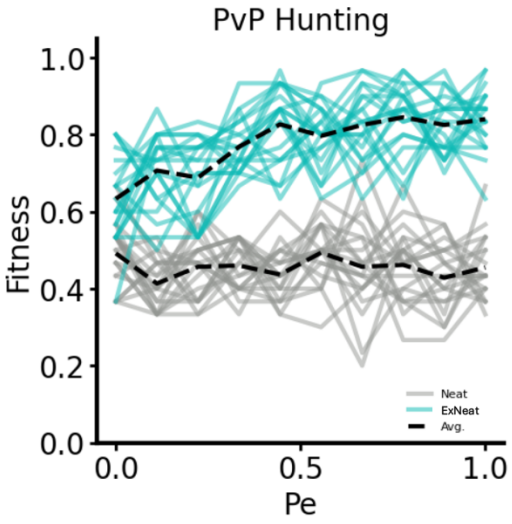


(a) Robustness against sensor noise scale variation.

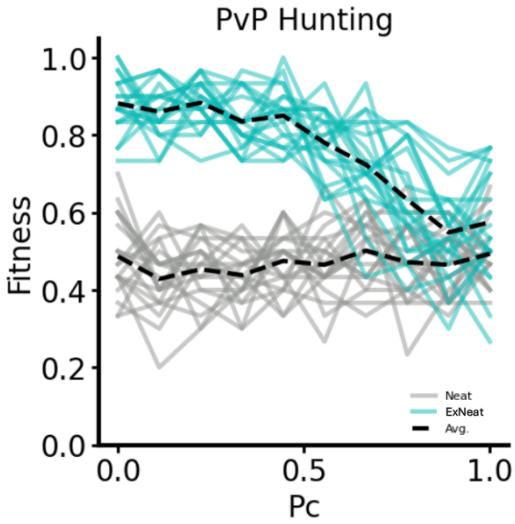
(b) Robustness against total prey number variation.

Figure 4.13: Testing performance comparison of agents in hunting task environment (1).

The results from both types of predator-vs-prey experiments showed that *ExNEAT* agents not only excelled at single-purpose navigation tasks, but also achieved excellent performance in combined tasks. Hence, there have been enough evidence that *ExNEAT* is a more powerful algorithm than NEAT in terms of problem solving in 2D environment.

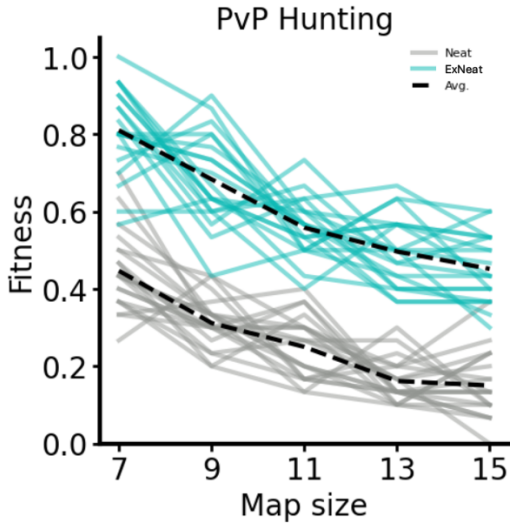


(a) Robustness against p_e variation.

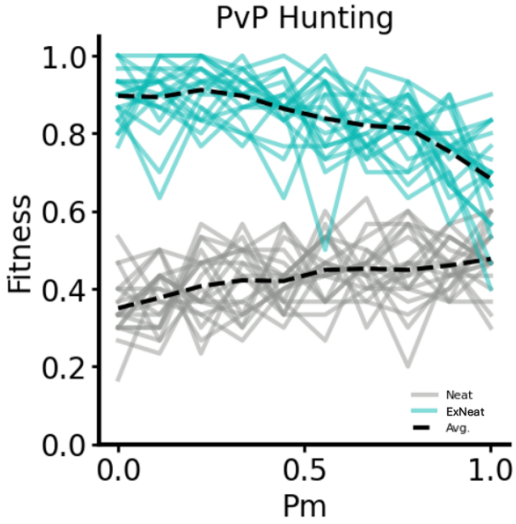


(b) Robustness against p_c variation.

Figure 4.14: Testing performance comparison of agent in hunting task environment (2). The fitness increases with the increase of p_e expectedly. And when p_c approaches 1, the fitness of *ExNEAT* drops to a low level. This is also normal since a p_c equals 1 means the trials emitted by the preys decays instantly. In practice, this means that the preys emits no trial at all.



(a) Robustness against environment size variation.



(b) Robustness against p_m variation.

Figure 4.15: Testing performance comparison of agent in hunting task environment (3). For NEAT agents, the performance increases slightly when Pm increases. But this is likely due to that NEAT agents are unable to properly locate the preys with its own decision making model and increase Pm increases the chance of the NEAT agent randomly encounter a prey.

4.6 Mujoco Ant experiment

4.6.1 Introduction

The maze navigation task and the predator-vs-prey experiment have both demonstrated the superiority of *ExNEAT* in assisting the learning of a single agent in a 2-dimensional Cartesian environment. However, despite that the aforementioned 2 experiments provided evidence for the potential of *ExNEAT*, the nature of the experiment may deem this conclusion to be less convincing. Specifically, the 2 experiments uses the simplest simulation strategy and some experiments, such as the 1D maze task can be theoretically completely even with trial and error. Moreover, the agents were assumed to be a single solid body and such system is unrealistic and would make a sim-to-real transfer difficult. Hence, we further tested the *ExNEAT* configuration by presenting it with a more challenging task. In particular, we will be stretching the potential of *ExNEAT* by training its agents to perform actions that involve physically realistic locomotion and dynamic control. Furthermore, during the previous experiments, the agents had identical modules being evolved for all 4 sensor locations. This simple configuration may work well for a majority of the RL tasks but it would be interesting to see the impact of evolving other configurations. Using more complex experiments will force the agent to have a more advanced network-module topology, such as recurrent connections between modules or having more than 4 modules.

4.6.2 Training results

For both NEAT and *ExNEAT*, we initialized populations with size 100. So there are 100 genomes groups in total. The NEAT agent has network with 6 hidden nodes with input size 105 and output size of 8. The *ExNEAT* agents used network with 4 hidden nodes with input size 22 and output size 2 for leg modules and input size 17 with output size 8 for the main body module. The maximum time steps the agents was allowed to attempt the task was set to 5000. During the training process, it is crucial to select the most suitable geometric configuration. Apart from the ”ALL” configuration, which guaranteed the discovery of the optimal geometry, all other configuration undergone an initial testing. And it was found that the ”FB”, ”LR” and ”Cross” configurations are unable to solve the task properly. Therefore, these configurations were not used in the proper training. Firstly, the ”Single” agent class was trained and assessed since this agent category has been proven to function well in the previous tasks. And then, we trained and tested the ”ALL”

configuration to see what difference changing the module combination/symmetry would have on performance.

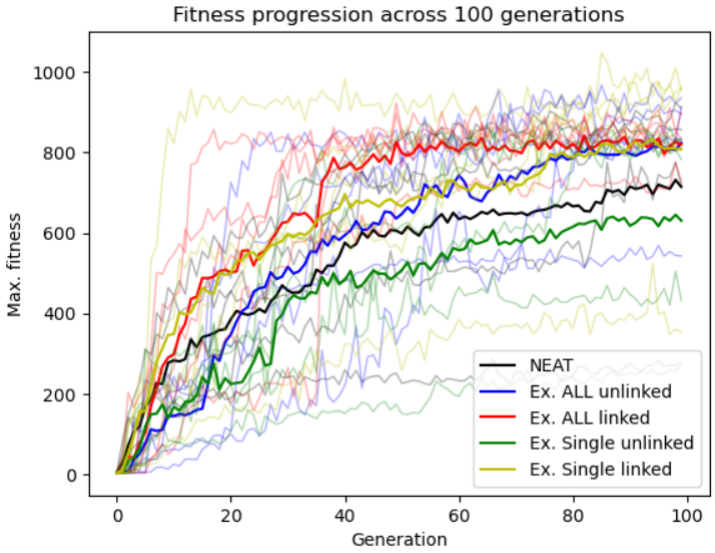


Figure 4.16: Training performance of NEAT vs 4 other best Modular NEAT configurations. The average fitness curve for each agent is highlighted with a thicker line.

The agents are trained with 100 generations. Each generation tests the fitness score of all 100 genomes groups once. For every generation, the genome groups that performed better than 90% of the current best genome were stored. We then plot the best fitness achieved for each generation as well as their average. (Figure 4.16).

Overall, adding interconnections between the modules was proved to improve training fitness in terms of the speed of convergence. This can be manifested from Figure 4.16 that, the unlinked "Single" configuration agents (green) generally performed worse than NEAT but the linked "Single" configuration (red) increased the fitness level to the same as the "ALL" configurations. The reason why the unlinked "Single" agents performed well for the maze solving and predator-vs-prey task but not in the Ant environment demonstrated the importance of sharing state information between modules in control of dynamic agents.

The "ALL" configuration has the freedom to evolve into any geometrical pattern, but it was expected to take longer because it needs time to discover the optimal pattern and reach the same fitness level as the "Single" configuration. Moreover, using a 5th module greatly helps with the training stages as an additional central block can utilize the compressed inputs from the 4 leg modules to discover more efficient solutions faster. Having a central module also helps with stabilizing the control as 4 modules with minimal communications may struggle to integrate together as a whole. The "Single" geometry configuration needs interconnected modules to retain

performance. This highlights the importance of how communication between the modules can also enforce symmetry and improve training stability.

From the training results, it can be concluded that, on average the majority of the *ExNEAT* agents offer better training performance than NEAT agents. The *ExNEAT* agents also achieved a better final fitness. Most of the *ExNEAT* agents also trained faster, i.e. required less generations to achieve the same level of performance as the NEAT agents.

4.6.3 Robustness

Robustness to initial disturbance

Given the training results in the previous section, it is expected that the training performance will be reflected in the testing performance. However, comparing to the previous two experiments, the Ant experiment offers a much higher degree of variation in terms of environment uncertainty. Therefore, it is hard to determine whether the trained agents would be robust against external disturbance. A series of tests will be required to completely verify the performance of *ExNEAT* agents.

During the testing process, the fitness score of each agent were evaluated 10 times for each generation to compute an average score, so that their fitness score plot should not fluctuate. Despite that, it is predictable for the test performance to be slightly worse. This is because the amount of environment that can possibly be explored by the agent is still less than the total possible search space. As the Ant agent was initialized with a random input vector of 105 dimension, a slight generalization error is likely to occur.

For the Ant environment, there are two important parameters that influence the agent's behavior: the reset noise scale and the sensor noise scale. The first parameter is related to the starting states of the agents. Namely, the initial position, velocity, torque and contact forces. Adding an initial perturbation to the system's states will affect the controllability of the Ant robot. Increasing the reset noise scale will test a neural network's ability to stabilize the agent. The second parameter is a continuous disturbance noise that is applied to the input data throughout the time steps. This noise type will instead assess the robot's ability to complete the task with incomplete or corrupted information, as would usually occur in a real life environment.

Another important, non-parametric assessment criterion is how well the agents can generalize to

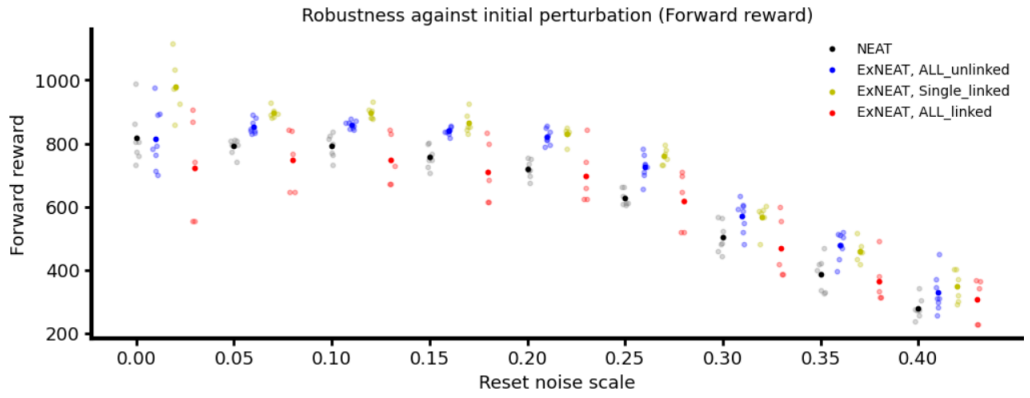


Figure 4.17: Robustness to reset noise. For each noise scale, we plot the fitness of each agent class and their mean as a cluster for easy visualization. In general, good training performance translates to good testing performance. One exception being that the linked "ALL" configuration performs less well in testing than in training. This is possibly caused by the fact that adding interconnections may disrupt symmetry during the training process. This means that the agent was able to discover better networks during training process but was unable to fully optimize it before training ends. Whereas in the "Single" configuration, the agents were forced to evolve symmetrical structure, which maintains a stable symmetrical structure throughout the training process.

achieve a good full reward. The reason why this measurement is crucial in performance evaluation is because the agents were trained using the forward reward, which omitted the influence of contact force and movement degree. The best performing Ant agents should be able to achieve a decent full reward after learning to walk using the forward reward.

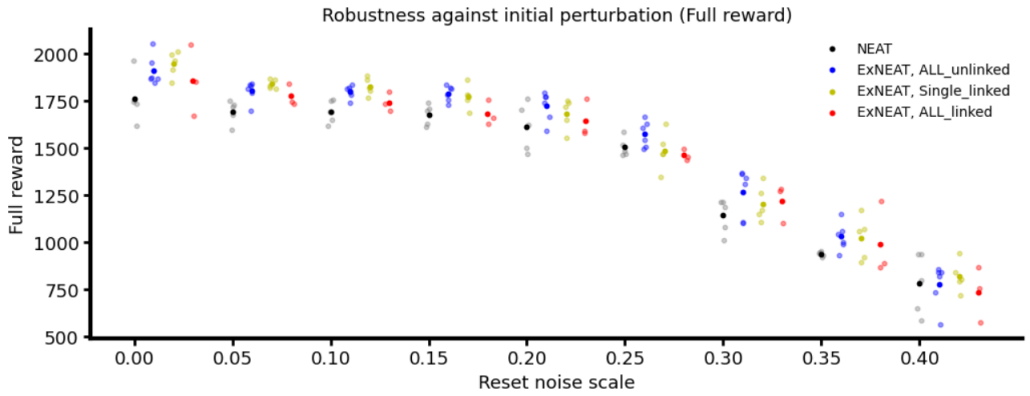


Figure 4.18: Robustness to reset noise scale and tested on achieving the highest full reward. In general, the performance of each agent is similar to the forward reward case, in which *ExNEAT* agents demonstrated good generalization than NEAT agents.

Robustness to sensor noise

The fact that *ExNEAT* agents generalize well under reset noise variation only means that they are better at stabilizing the system from an initial state. To fully confirm the robustness of the agents, we must assess their behavior under continuous variations. Hence, the second part of the testing

focuses on testing the agent's robustness against the sensor noise. As previously mentioned, the sensor noise is a continuous perturbation to the sensing information such as current position and velocity. This variable will have a strong influence on the agent's decision making process. Since decision making is a crucial part in navigation, a good agent should be able to resist a certain amount of sensor data corruption.

In our experiment, we implemented the sensor noise by adding an array of normally distributed noise with a standard deviation to the observation space. The sensor noise scale is therefore equivalent to the standard deviation of the noise. The results showed that the performance of NEAT agents decreases quickly once the sensor noise level starts to increase. On the contrary, ExNEAT agents are almost insusceptible to noise. This demonstrated how a modular layout effectively averaged out the effect of noise and allowed the agent to maintain a stable control. For a single network configured model such as a NEAT agent, a small local influence within the network is directly passed onto the entire network. Hence, the output can become corrupted easily. Whereas for the *ExNEAT* model, the effect of noise was first filtered by the leg modules, and the compressed input by the leg modules meant that the central module will receive a relatively uncontaminated signal.

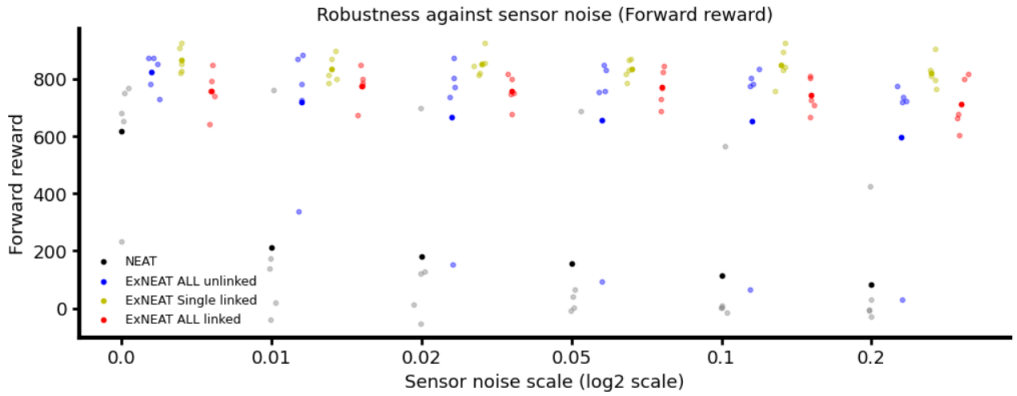


Figure 4.19: Robustness to sensor noise. Same as before, we chose 5 top agents from each agent classes and performed 100 repeats to estimate the average results.

Similarly, the agent's ability to generalize to the full reward was also tested for the sensor noise case. On average, the reward achieved by the NEAT agents decreases rapidly as sensor noise level increases; the *ExNEAT* agents on the other hand, remain unaffected.

It was also found that the performance of *ExNEAT* agents only dropped significantly when the noise level reaches a standard deviation of 2. The relevant figures can be found in the appendix.

Overall, the *ExNEAT* agents out-performs NEAT in training and are more robust to testing.

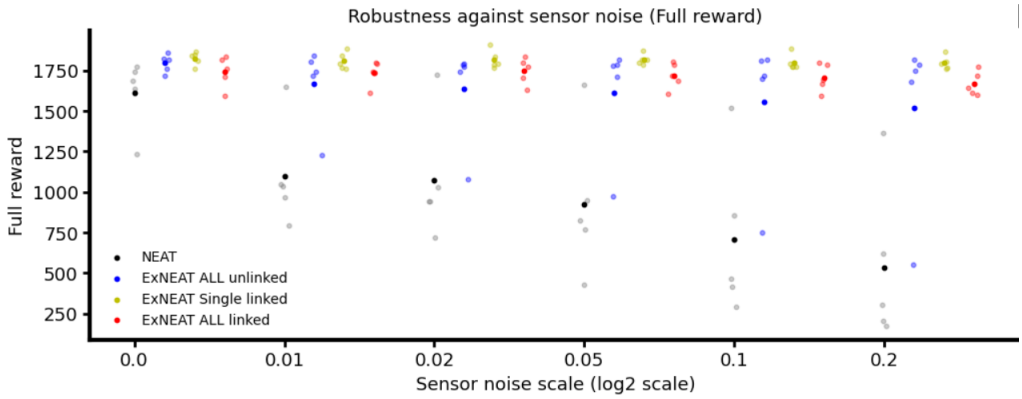


Figure 4.20: Robustness to sensor noise (full reward).

Moreover, it is extremely robust to sensor noise. These results provided evidence that the *ExNEAT* agents are more superior in all 1D, 2D and 3D tasks. Moreover, *ExNEAT* agents have been shown to offer a better performance in both discrete navigation tasks and continuous, dynamic robotic control tasks.

5 Evaluation

Contents

5.1 Evaluation plan

The methodology which is used to gather the experimental results obtained so far can be reused for any other future experiments. The future models would have a greater complexity and any future experiments would also involve more agents and freedom of exploration. But the main steps of evaluating the agents should not differ by too much. The experimental results should justify the use of *ExNEAT* as a better alternative to other agent types for reinforcement learning tasks and provide enough insights by means of graphs or diagrams to further demonstrate the advantages and superiority of this novel neuroevolution algorithm. We tested the following configurations: 4 different modules unconnected, front-back symmetrical modules interconnected and 4 different

modules interconnected. These 3 configurations are the best performing ones according to their training data. The front-back symmetrical unconnected configuration was omitted as it was evident that it will not perform well enough.

In terms of evaluation of target achievements, various goals can be set to ensure the project continues as smooth as possible. For example, for every new method being devised, a new set of experiments will be carried out and results documented to measure the effectiveness. Even the smallest achievements, such as improving the display of a certain figure or optimizing the speed of certain parts of the code can be used as milestones for the project's progression. These, combined with larger milestones such as finishing a certain parts of the project before the end of each term, will make sure that progress is always being carefully monitored.

5.2 Performance comparison

The overall performance was compared using the best average fitness achieved by each agent, labeled as *fitness* and the efficiency of the agent, which is how much generation each agents need in order to reach the same level of fitness.

5.2.1 Maze experiment

	Noise=0.05		Size=17		Gap=2	
Agent	f_{train}	efficiency	$f_{fitness]}$	efficiency	$f_{fitness}$	efficiency
NEAT	78.69%	20+ gen.	90.18%	13 gen.	58.98%	20+ gen.
ExNEAT	93.48%	7 gen.	100%	3 gen.	100%	7 gen.

Table 5.1: 1D track maze performance comparison.

	Noise=0.05		Size=11	
Agent	f_{train}	efficiency	f_{train}	efficiency
NEAT	30.70%	18 gen.	38.71%	20 gen.
ExNEAT	53.31%	2 gen.	95.60%	2 gen.

Table 5.2: 2D general maze performance comparison.

5.2.2 Predator vs prey experiment

Each fitness results were computed with the average performance of 10 agents each with 20 repeats of maze trials. So a total of 200 repeats per generation is used to estimate the performance.

	Noise=0.01		pc=0.25	
Agent	f_{train}	efficiency	f_{train}	efficiency
NEAT	43.27%	20+ gen.	43.73%	13 gen.
ExExNEAT	99.33%	1 gen.	99.33%	3 gen.

Table 5.3: Foraging performance comparison.

	Noise=0.01		pc=0.25		pm0.5	
Agent	f_{train}	efficiency	f_{train}	efficiency	f_{train}	efficiency
NEAT	53.00%	6 gen.	54.67%	20+ gen.	53.73%	14 gen.
ExExNEAT	94.61%	1 gen.	96.89%	1 gen.	98.11%	1 gen.

Table 5.4: Hunting performance comparison.

ExNEAT managed to perform better in all cases. And in several scenarios, the performance level of *ExNEAT* is approximately double that of NEAT. In terms of training efficiency, *ExNEAT* generally requires 4-10+ fold less generations to reach the same performance as NEAT.

5.2.3 Mujoco Ant experiment

We denote f as the forward test reward and F the full reward that took into account the contact forces and movement degree. The table below displays the agent's training data, the training efficiency as well as the agents' robustness against initial noise disturbance n_i and sensor noise level n_s. The efficiency is measured relative to the NEAT agent's performance. It is the number of generations that *ExNEAT* required to achieve the same level of fitness as NEAT. The initial noise level and the sensor noise level used for testing was set to 0.2.

	NEAT	Ex. ALL unlinked	Ex. Single linked	Ex. ALL linked
f_{train}	732	823	824	842
efficiency	98 gen.	69 gen.	57 gen.	37 gen.
$f_{test,n_i=0.2}$	720	821	829	698
$F_{test,n_i=0.2}$	1612	1723	1683	1645
$f_{test,n_s=0.2}$	720	821	829	698
$F_{test,n_s=0.2}$	532	1767	1798	1667

Table 5.5: Mujoco Ant performance comparison. The agents were trained for 10 repeats and the average of 5 agents are used to estimate an average performance. In total 50 repeats for each generations. For the testing experiments, the 5 top performing agents from each category is used to estimate the performance. Each agents were evaluated for 100 trials and a total of 500 repeats for each sweep point.

The ExNEAT agents are expected to perform better in the testing environment since they also have a better training performance. An exception being the ExNEAT ALL linked configuration, which has a slightly lower forward fitness. This shows that adding extra connections may lead to overfitting. For the generalization of full reward testing, all *ExNEAT* agents have a better average performance. This demonstrated that a modular configuration produces agents that are

more robust in terms of control forces and stability.

5.3 Evolution trials

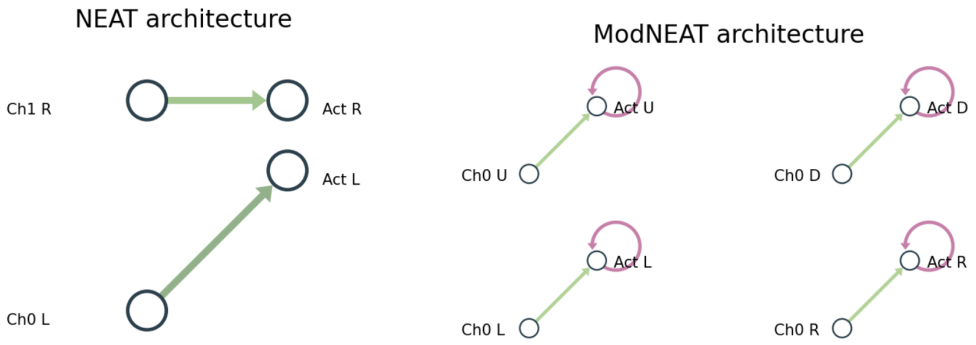
5.3.1 Introduction

Apart from achieving a higher fitness score in the proposed experiments, this project also aims to conduct further research into the evolution process of a multi-module system and to study the reason why it is able to out-perform traditional NEAT. In particular, we studied the architecture property and how it influences the behavior of the agents.

To visualize the agent's model network, they have been plotted as netgraphs where a circle indicates a *neuron* and the coloured arrows indicates the *weights* between 2 neurons. Neurons to the left and right most of the diagram and the input and output neurons respectively and have been labeled accordingly.

5.3.2 1D track maze

Scenario 1: normal condition



(a) A typical NEAT architecture solution

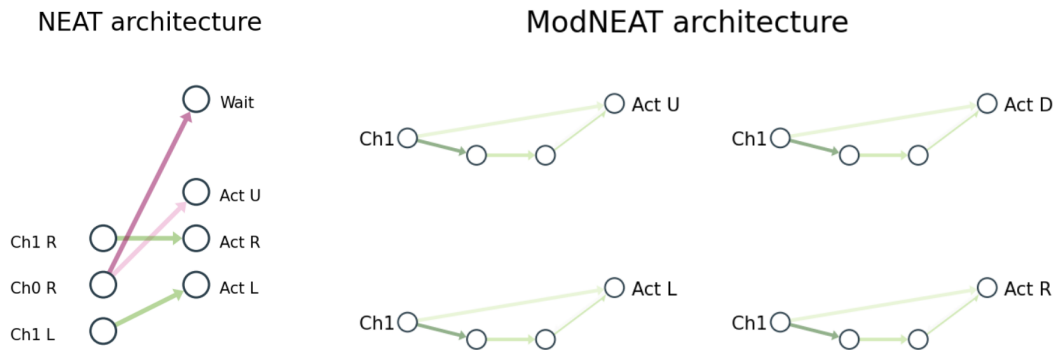
(b) A typical *ExNEAT* architecture solution

Figure 5.1: Comparison of architectural difference between a NEAT agent and a *ExNEAT* agent. Due to the symmetrical configuration, it can be observed that the weights are equal across all 4 sensors for the *ExNEAT* agents. While the NEAT agent's network evolved different weights for each sensor. This explains why NEAT may struggle to choose between left and right direction and why *ExNEAT* agents are more likely to make the correct decision.

As stated in the previous sections, track maze includes a gap period in time where the agent would receive no inputs (see Figure 3.6). This requires the agent to evolve the ability to memorize

previous steps taken using recurrent connections. As can be seen here, *ExNEAT* has a set of 4 modules each with recurrent connection at the output neuron whereas NEAT failed evolve a meaningful memory function.

Scenario 2: noisy condition

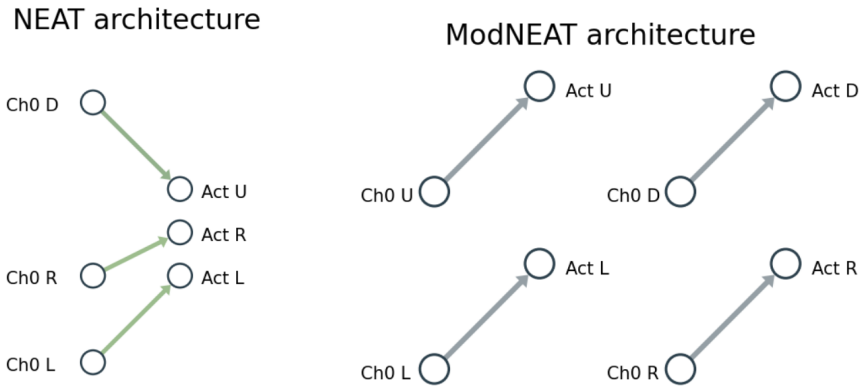


(a) A typical NEAT architecture solution

(b) A typical *ExNEAT* architecture solution

Figure 5.2: In a noisy condition, it is expected for a model to develop a denoising mechanism in order to filter out incorrect information. In this case, *ExNEAT* has managed to evolve a 3 layer strategy which acts both as a memory storing system and a denoising network whereas NEAT fails to evolve either of the desired trial to adapt to the environment.

5.3.3 2D general maze

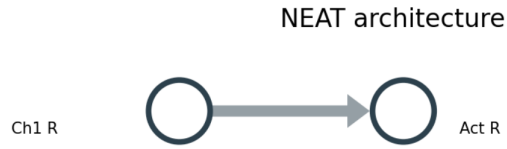


(a) A typical NEAT architecture solution

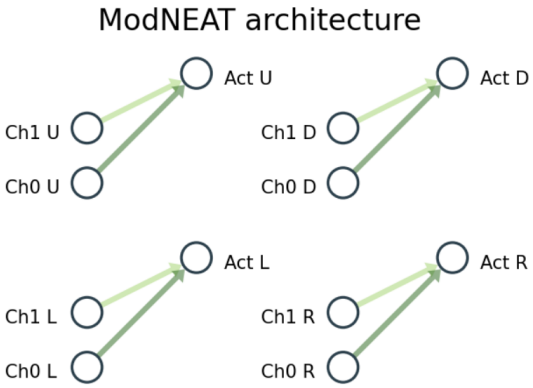
(b) A typical ExNEAT architecture solution

Figure 5.3: In an environment with 4 direction of movement, it's crucial for the agent to develop sensor for each locations (up, down, left and right). However, for NEAT, some sensor locations are incorrectly discarded, leading to misinformation and loss of performance. The 4 module configuration of *ExNEAT* on the other hand, ensures accurate spatial perception with a minimal model size.

5.3.4 Predator vs prey



(a) A typical NEAT architecture solution



(b) A typical *ExNEAT* architecture solution

Figure 5.4: A similar trial can be identified through the predator-vs-prey experiment where all 4 sensor directions are important for navigation. For NEAT, a majority of the sensor locations are incorrectly discarded, leading to misinformation and loss of performance. The 4 module configuration of *ExNEAT* on the other hand, ensures accurate spatial perception with a minimal model size.

5.3.5 Mujoco Ant

We first looked at the architecture trial produces by a typical neat agent (Figure 5.5). The architecture was chosen from one of the best performing neat genomes as it is a good representation of any top performing neat agents. Next, the architecture for the ExNEAT agent are plotted to draw a comparison with NEAT (Figure 5.5-5.7). Specifically, the *ExNEAT* agents trained with "All" configuration and the ones trained with "Single" configuration. Including the leg and main module.

The "Single" module configuration has exhibited outstanding performance for the maze solving and predator-vs-prey task. Hence, it would be expected to perform well in the Ant environment. The fact that all 4 legs are identical provided a strong overall symmetry (both line symmetry and rotational symmetry). Making the system more robust in terms of control stability and decision making process. Another advantage is that it produces less redundant neurons than a NEAT agent's network. Moreover, because the leg modules have pre-processed the input data, the central

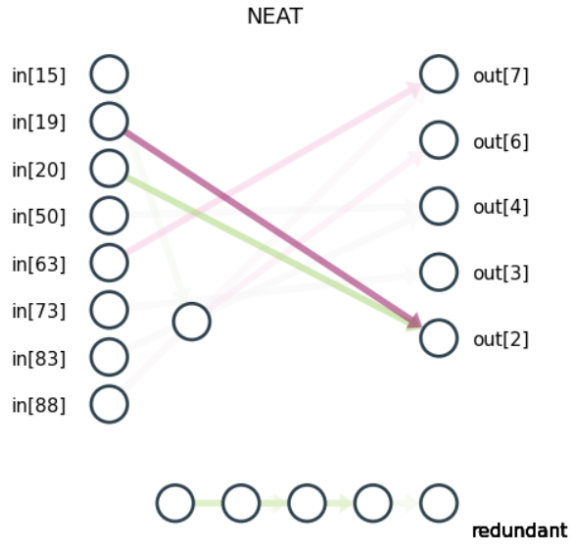
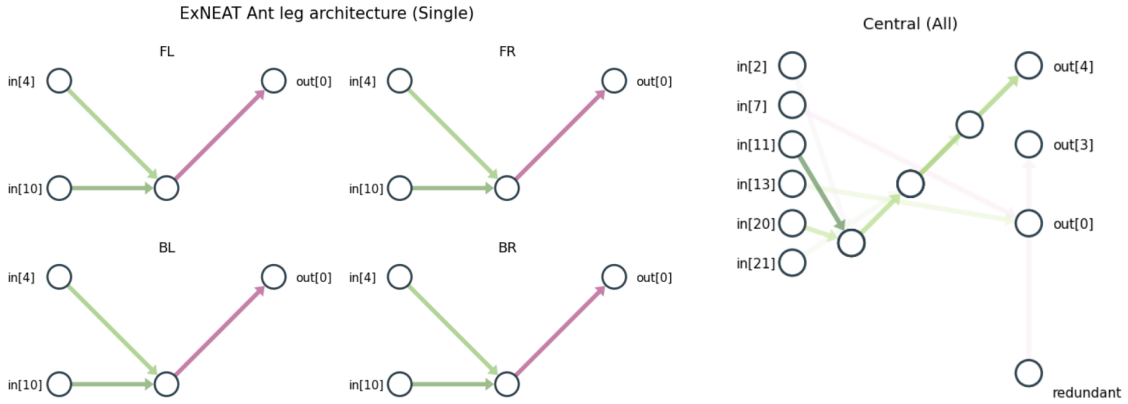


Figure 5.5: A typical NEAT architecture evolved in the Mujoco Ant environment. Despite the presence of multiple input and output neurons, most of their connection weights are negligible and only 2 weight connections are significant. Moreover, despite that NEAT was given the opportunity to evolve 6 hidden nodes, only 1 hidden node was utilized and 5 of them becomes redundant node.



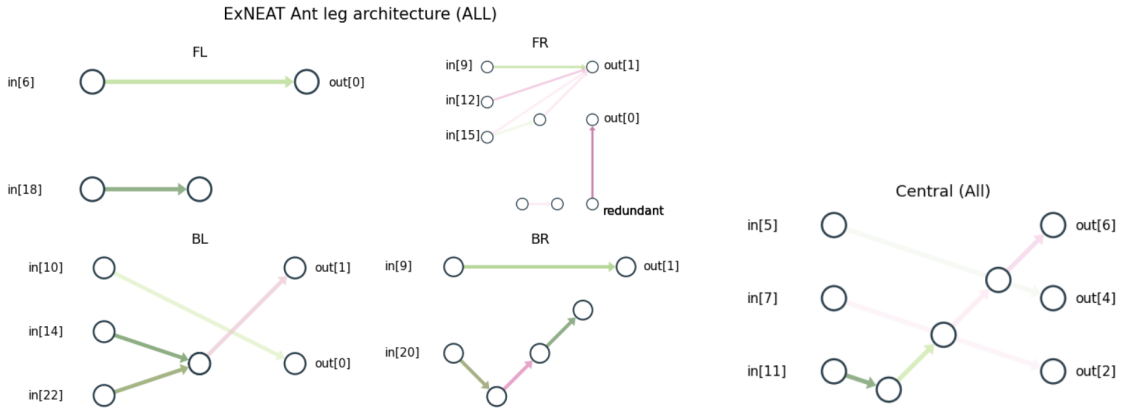
(a) 4 leg architectures.

(b) Main body architecture.

Figure 5.6: A typical ExNEAT architecture with "Single" configuration evolved in the Mujoco Ant environment.

module can now utilize a much fewer number of input and output nodes and still offers a better performance than NEAT. This shows that *ExNEAT* algorithm tends to produce solutions that are more simple and efficient.

The "All" configuration allows the leg agent to evolve into different architectures. Each one can be formed depending on the particular task that was local to that module, such as sensing, control or movement. From Figure 5.7, it can be seen that some architectures, such as the back-left (BL), back-right (BR) leg module and the central module evolved longer chains to optimize memory and denoising. Some architectures, such as the front-right (FR) modules focused on decision making by



(a) 4 leg architectures.

(b) Main body architecture.

Figure 5.7: A typical ExNEAT architecture with "All" configuration evolved in the Mujoco Ant environment.

averaging inputs from many nodes. While some architectures, such as the front-left (FL) module consists of simple structure that usually makes a discrete or binary decision, such as all or nothing. The diversity of the leg modules is one of the key factor why ExNEAT agents are able to discover more optimal and more importantly, more efficient solutions. A better efficiency ultimately implies better generalization and more robust to variations.

5.3.6 Summary

From the comparison of both the results sections and the evaluation section, it is evident that ExNEAT is robust in terms of training performance, test robustness and evolution efficiency. The idea of predefining sensor substrate ensures the high-level structure of an agent is not disrupted during evolution while the lower-level, more varied features can be added for advanced functionalities. This ultimately leads to the drastic increase in performance as well as the stability in both training and testing environment. Most importantly, we have demonstrated that *ExNEAT* are able to solve a wider variety of tasks than the previous *ModularNEAT* [45]. Furthermore, we have demonstrated that *ExNEAT*, like HyperNEAT offers the ability to evolve custom geometric topologies to suit the environment. But it does not require to evolve an extra group of neural networks. Overall, we have successfully demonstrated the advantages of *NEAT* over NEAT in control and navigation problems in: (1) all 1D, 2D and 3D situations. (2) in both discrete and continuous environments. (3) with agents being static or dynamic. (4) with single or multi-body robotic systems. We have also studied the implications behind why *ExNEAT* offers a significant improvement in the evolution process in terms of the topologies of the networks.

6 Reflection

Contents

6.1 Communication

This project is heavily centered in modern artificial intelligence (AI) and machine learning (ML). Many relevant sources and concepts require technical knowledge to be understood. Therefore, the report and its associated presentation slides have been carefully worded and logically structured to ensure that people who are unfamiliar with the work will be able to easily follow through. Specifically, many key concepts have been explained using simple terms during their first appearance so that the reader can easily refer back to check their understanding of the subject. Moreover, a selection of elaborately drawn diagrams with clear instructions and intuitive illustrations will enable anyone to easily comprehend, at a high-level, the main concepts of the project.

The effectiveness of communication can be evaluated by having the report reviewed by the supervisors and specific feedback obtained about writing will be noted. From the interim report feedback there has been no issues with the way of communication or causes for ambiguities in any sections of the report, thus proving this piece of writing is at least clearly represented for technical audiences. Moreover, this project will be used further in its associated laboratories, such as the intelligent systems lab or the personal robotic lab. Having the work done clearly documented in

this thesis will allow newcomers to quickly settle themselves into the project and start making progress. If this project is published as a research paper, it will also allow reader to quickly draw useful information through this report. Overall, any usage of this report would create opportunity for feedback and evaluation.

6.2 Environmental and Societal impact

The purpose of this project is to discover new methods in robotic navigation in complex environment. Any potential application of this research will be used to assist human operations and improve the trustworthiness of autonomous machines. This project should also be carried out in a fully simulated environment hence no physical equipment will be present for field testing. There is no plan to use any licensed software or technology and any code base is fully open-source and free to use.

Any potential cause of ethical or social issue has been taken into account and listed here:

1. Animal or human body trials: No living organisms are used in this project so no related concerns shall arise.

2. Societal damage through misuse: The purpose of the project is to train robots that perform navigation tasks and any trained agents cannot perform tasks beyond what it was trained for. Only trusted researchers will have access to the source code that were used to train the agents thus ensures the security when using the project's developing tools or packages. The project's code are stored safely in GitHub to prevent malicious access.

3. Environmental damage: The only possible environmental influence this project could have would be the training energy costs. As commonly known, deep learning hardware consumes larger amounts of energy than traditional machine learning and can indirectly contribute to environmental damage. Because of this, the training environments of this project are chosen to only run on CPU, which is the most power efficient and requires the least resources to operate.

Conclusions

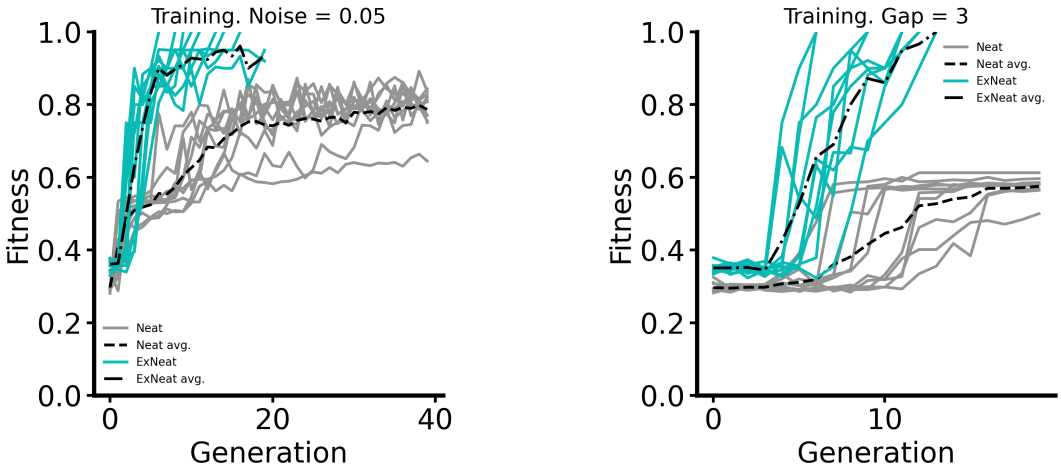
In this project, we have presented *ExNEAT*, a neuroevolution algorithm based on the NEAT algorithm. The model offers tremendous improvement in solving navigation problems in terms of reducing the search space, saving memory and requiring less training generations. We have also shown that it can also out-perform NEAT in the control and navigation of a physically complex, real-life transferable robotic system. Most importantly, *ExNEAT* took inspiration from the symmetrical property of biological brains and applied it to the maze solving task. A significant change to traditional NEAT is the design of a new evolution algorithm that enabled coevolution of multiple networks. The process of assimilating the Python implementation of NEAT, analyzing its low-level functions and combining it with custom algorithms was a crucial but challenging step. However, the result is that it added more degree of freedom and a more suitable evolution process that is tailored for modular-based network evolution. The techniques used to re-design NEAT's evolution cycle have also improved competency in deep learning and programming, which is a crucial step to become proficient at neuroevolution research. In terms of application, this pipeline can be a useful application in household robots or navigation robots such as self-driving vehicles or quadruped field operation robots. One of the most significant discovery was that the modular networking turned out to be a powerful approach which enables simple scalability, and it can be developed further to suit a wide range of scenarios. Furthermore, the fact that *ExNEAT* can easily generalize to different task scenarios provides evidence that the potential of ExNEAT is not limited to its own domain. Some of the other potential applications of *ExNEAT* could include image processing or language models. Therefore, this new algorithm will open up numerous possibilities for any future research endeavors.

7 Appendix

Contents

7.1 Appendix A - Additional 1D Maze training comparison

This section documented the different aspects of the training history of *NEAT* and *ExNEAT* in solve the 1D maze navigation problem. The agents are trained under conditions that: (1) has high noise. (2) Contains temporal missing data. (3) Is excessively complex. For each aspects the experiment is repeated 10 times to produce a more accurate representation of the agent's training performance.



(a) 1D maze. Trained under high noise environment.

(b) 1D maze. Trained with temporal missing data.

Figure 7.1: Testing performance comparison of agents under reset noise variation.

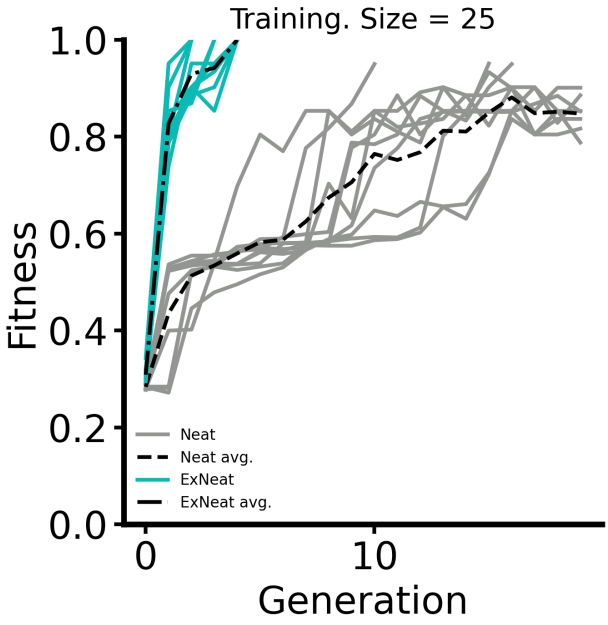


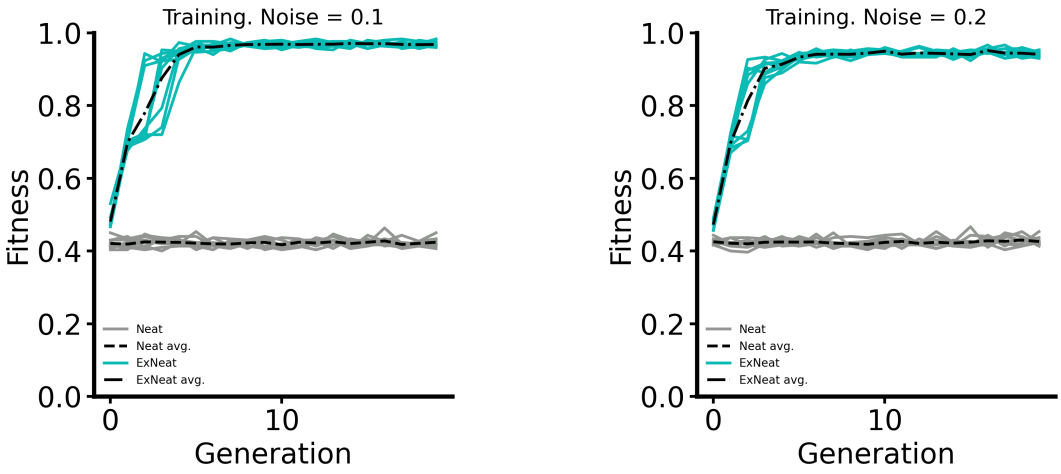
Figure 7.2: 1D maze. Trained with complex mazes.

7.2 Appendix B - Additional training comparison of predator-vs-prey experiment

This section documents the training performance of *ExNEAT* and NEAT agents under adverse conditions.

7.2.1 Foraging

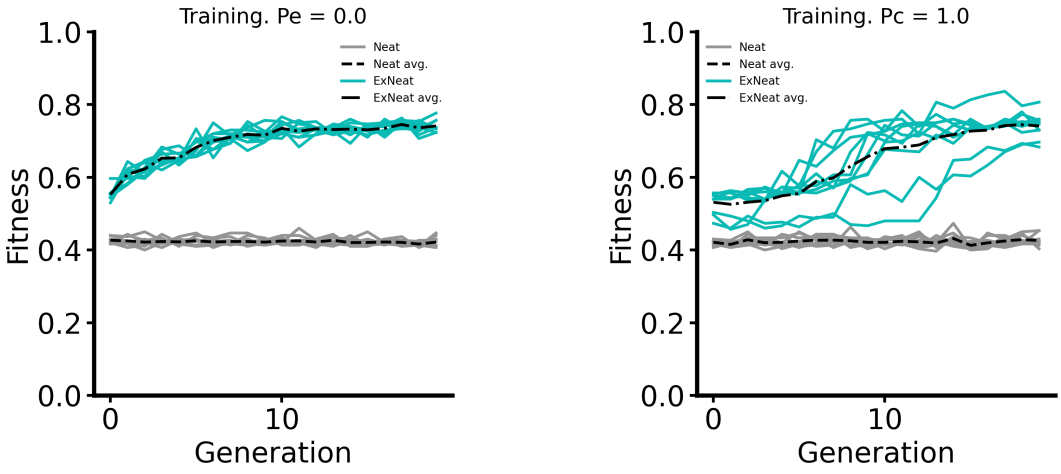
Additional training results for the foraging experiment.



(a) Trained under high noise environment (1).

(b) Trained under high noise environment (2)

Figure 7.3: Testing performance comparison of foraging task agents under extreme environments (1).



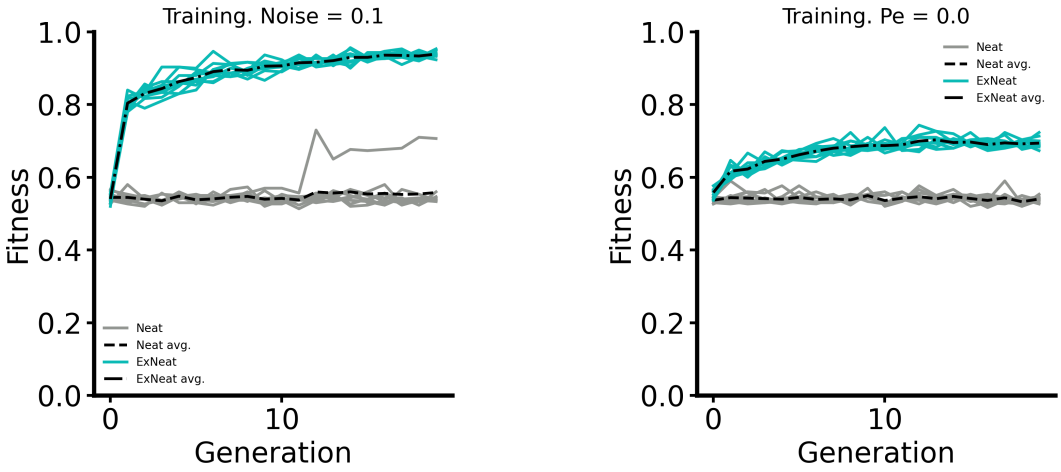
(a) Training with preys that does not emit sensory cues (1).

(b) Training with instant decay of sensory cues (2)

Figure 7.4: Testing performance comparison of foraging task agents under extreme environments (2).

7.2.2 Hunting

Additional training results for the hunting experiment.



(a) Trained under high noise environment.

(b) Training with preys that does not emit sensory cues.

Figure 7.5: Testing performance comparison of hunting task agents under extreme environments.

7.3 Appendix C - cluster plots for the agents' training results in Mujoco Ant environment

As mentioned in section 4.6.2, one may be interested in observing and comparing the overall performance of each agents. The following plot shows the average training performance of each agent across 100 generations (found by calculating the area under each curve in Figure 4.16).

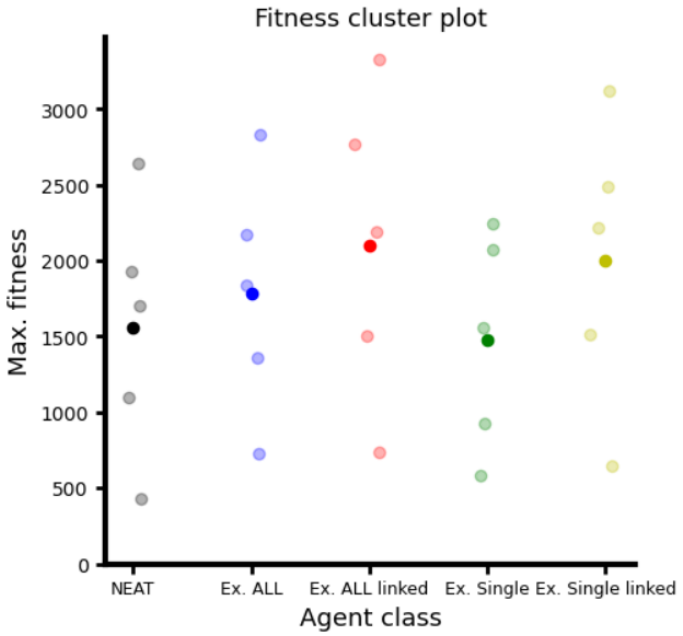
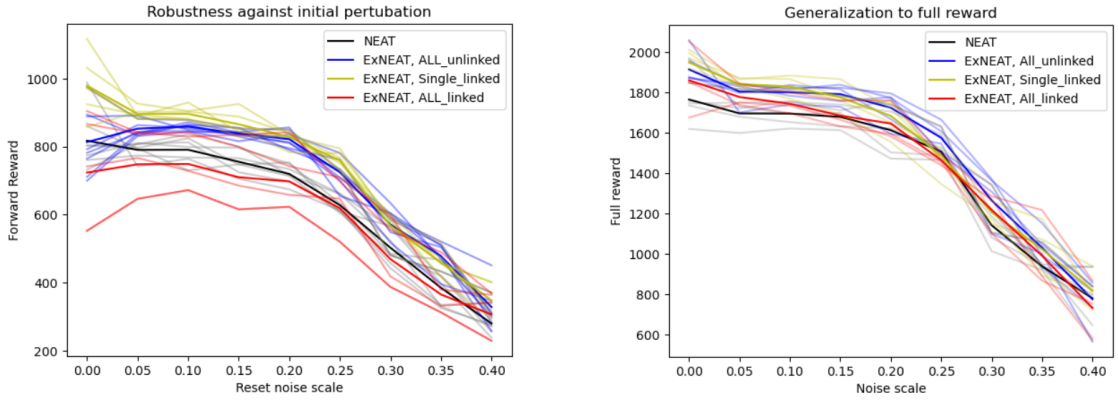


Figure 7.6: Cluster plot for the training performance of each agent.

7.4 Appendix D - line plots for the Mujoco Ant environment testing experiments

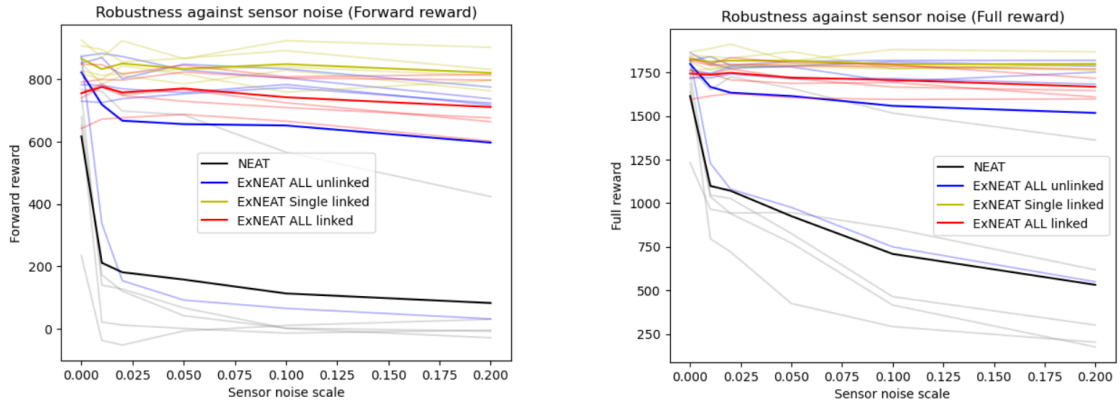
The results figures in the section 4.6.3 displays the robustness measurement of each agent class as a cluster plot with mean and spread clearly illustrated. This was to allow easy comparison between the agents. One might find it useful to also observe the line plots of the agents. The line figures are less readable but can provide important observation such as the trend of the agent's robustness and the local variations about a certain sweep point.



(a) Forward reward testing performance.

(b) Full reward testing performance.

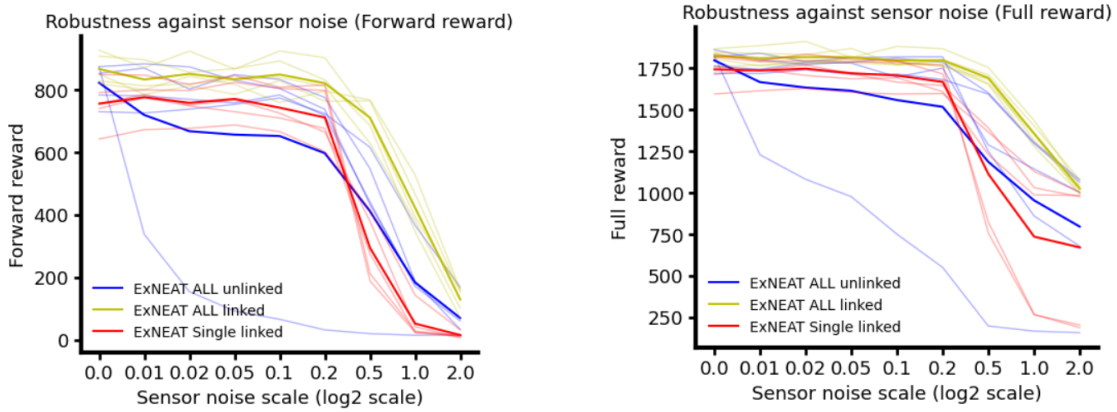
Figure 7.7: Testing performance comparison of agents under reset noise variation.



(a) Forward reward testing performance.

(b) Full reward testing performance.

Figure 7.8: Testing performance comparison of agents under sensor noise variation.



(a) Forward reward testing performance.

(b) Full reward testing performance.

Figure 7.9: Further verification of *ExNEAT* agent's robustness against sensor noise scale. The sensor noise variance is extended to 2.

Bibliography

[1] M. R. Stanley KO, "Evolving neural networks through augmenting topologie," *Evol Comput*, vol. 10, pp. 99–127, 2002.

[2] P. Mickle, "A peep into the automated future," *The capital century 1900–1999. http://www. capitalcentury. com/1961. html*, 1961.

[3] P.-B. Wieber, R. Tedrake, and S. Kuindersma, "Modeling and control of legged robots," in *Springer handbook of robotics*, Springer, 2016, pp. 1203–1234.

[4] J. Liu, M. Tan, and X. Zhao, "Legged robots—an overview," *Transactions of the Institute of Measurement and Control*, vol. 29, no. 2, pp. 185–202, 2007.

[5] G. Vachtsevanos, K. Davey, and K.-M. Lee, "Development of a novel intelligent robotic manipulator," *IEEE Control Systems Magazine*, vol. 7, no. 3, pp. 9–15, 1987.

[6] S. G. Tzafestas, "Mobile robot control and navigation: A global overview," *Journal of Intelligent & Robotic Systems*, vol. 91, pp. 35–58, 2018.

[7] F. Gul, W. Rahiman, and S. S. Nazli Alhady, "A comprehensive study for robot navigation techniques," *Cogent Engineering*, vol. 6, no. 1, p. 1 632 046, 2019.

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[9] J. Brownlee, "The pole balancing problem: A benchmark control theory problem," 2005.

[10] K. Doya, "Reinforcement learning: Computational theory and biological mechanisms," *HFSP journal*, vol. 1, no. 1, p. 30, 2007.

[11] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[12] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," *Technical Report CUED/F-INFENG/TR 166*, Nov. 1994.

[13] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 1, Jan. 2021, ISSN: 1532-4435.

[14] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, pp. 3–52, 2002.

[15] R. Ingo, "Evolution strategy: Optimization of technical systems by means of biological evolution. fromman-holzboog," *Stuttgart*, vol. 104, p. 15, 1973.

[16] G. B. Fogel and D. B. Fogel, "Continuous evolutionary programming: Analysis and experiments," *Cybernetics and System*, vol. 26, no. 1, pp. 79–90, 1995.

[17] D. B. Fogel, "An overview of evolutionary programming," in *Evolutionary algorithms*, Springer, 1999, pp. 89–109.

[18] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm-a literature review," in *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*, IEEE, 2019, pp. 380–384.

[19] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[20] 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.

[21] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 6, pp. 476–493, 2021.

[22] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 1, pp. 25–41, 2015.

[23] R. Akut and S. Kulkarni, "Neuroevolution: Using genetic algorithm for optimal design of deep learning models," in *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, IEEE, 2019, pp. 1–6.

[24] e. a. Such Felipe Petroski, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning.," *arXiv preprint*, vol. arXiv:1712.06567, 2017.

[25] e. a. Salimans Tim, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint*, vol. arXiv:1703.03864, 2017.

[26] A. Nelson, E. Grant, J. Galeotti, and S. Rhody, "Maze exploration behaviors using an integrated evolutionary robotics environment," *Robotics and Autonomous Systems*, vol. 46, no. 3, pp. 159–173, 2004, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2003.11.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889003001830.

[27] F. J. Gomez, R. Miikkulainen, *et al.*, "Solving non-markovian control tasks with neuroevolution," in *IJCAI*, Citeseer, vol. 99, 1999, pp. 1356–1361.

[28] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, pp. 341–359, 1997.

[29] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[30] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, ieee, vol. 4, 1995, pp. 1942–1948.

[31] R. Miikkulainen, J. Liang, E. Meyerson, *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*, Elsevier, 2024, pp. 269–287.

[32] K. Pretorius and N. Pillay, "Neural network crossover in genetic algorithms using genetic programming," *Genetic Programming and Evolvable Machines*, vol. 25, no. 1, p. 7, 2024.

[33] C. Igel, "Neuroevolution for reinforcement learning using evolution strategies," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, IEEE, vol. 4, 2003, pp. 2588–2595.

[34] M. Hiraga and K. Ohkura, "Topology and weight evolving artificial neural networks in cooperative transport by a robotic swarm," *Artificial Life and Robotics*, vol. 27, no. 2, pp. 324–332, 2022.

[35] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355–366, 2014.

[36] K. Vignesh Kumar, R. Sourav, C. Shunmuga Velayutham, and V. Balasubramanian, "Fitness function design for neuroevolution in goal-finding game environments," in *Advances in Computational Collective Intelligence: 12th International Conference, ICCCI 2020, Da Nang, Vietnam, November 30–December 3, 2020, Proceedings 12*, Springer, 2020, pp. 503–515.

[37] S. Rodzin, O. Rodzina, and L. Rodzina, "Neuroevolution: Problems, algorithms, and experiments," in *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*, IEEE, 2016, pp. 1–4.

[38] A. Furman, "Multi-robot systems, complexity and multi-objective neuroevolution in evolutionary robotics: A review,"

[39] P. Reyes and M.-J. Escobar, "Neuroevolutive algorithms for learning gaits in legged robots," *IEEE Access*, vol. 7, pp. 142 406–142 420, 2019.

[40] F. Tanaka and C. Aranha, "Co-evolving morphology and control of soft robots using a single genome," in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2022, pp. 1235–1242.

[41] A. Ram, G. Boone, R. Arkin, and M. Pearce, "Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation," *Adaptive behavior*, vol. 2, no. 3, pp. 277–305, 1994.

[42] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[43] D. E. Moriarty, *Symbiotic evolution of neural networks in sequential decision tasks*. The University of Texas at Austin, 1997.

[44] J. Buck, "Maze generation: Aldous-broder algorithm," *The Buckblog*, 2011. [Online]. Available: `https://weblog.jamisbuck.org/2011/1/17/maze-generation-aldous-broder-algorithm`.

[45] J. Reisinger, K. O. Stanley, and R. Miikkulainen, "Evolving reusable neural modules," in *Genetic and Evolutionary Computation – GECCO 2004*, K. Deb, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 69–81, ISBN: 978-3-540-24855-2.