Immuno-Inspired Embodied Lifelong Learning in Robots

Thesis submitted in partial fulfilment of the requirements
for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Divya D. Kulkarni

Under the supervision of

Prof. Shivashankar B. Nair

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Guwahati - 781039 Assam India

September, 2022

Dedicated to

My beloved Parents

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my thesis advisor Prof. Shivashankar B. Nair for his guidance, patience, and support throughout my doctoral research. I am forever indebted to him for turning me from an amateur research student to a mature researcher and grooming me to think uniquely. His encouragement and opportunities to participate in conferences and research talks and present the work in such venues have greatly improved my oratory skills. Also, his strong ethics and morals, which I have hopefully incorporated, will help me in my career and life.

I am grateful to Prof. Pradip K. Das, chairperson of my thesis doctoral committee, for his continual support and guidance. I also would like to thank Dr. Pinaki Mitra and Dr. Harshal Nemade for their astute inputs and suggestions, which helped me to improve the thesis work.

Words cannot express gratitude to my mother, Vidya Kulkarni, who always stood by me at all times, cheering and encouraging me. Her indomitable spirit and unwavering support has helped me achieve my goals. I am thankful to my father, Dattaraj Kulkarni, for his encouragement and patience. I miss my late uncle Ramachandra Dambal and my late grandmother Bharatibai Dambal. I will be forever indebted for their incredible support and wishes. My fiance, Paritosh Katyal, has been my biggest support and strength. I also thank my cousins Neha, Abhay, Rohan and Shruti. I am incredibly thankful to all my family members and friends for their unswerving support.

I want to thank the heads of the Department of Computer Science and Engineering during my Ph.D. at IIT Guwahati, Prof. Diganta Goswami, Prof. S V Rao, and Prof. Jatindra Kumar Deka, for letting me use the facilities of the department and for providing me the travel support to attend the conferences. I am thankful to Mr. Hemanta Kumar Nath, Mr. Pranjitt Talukdar, Mr. Raktajit Pathak, Mr. Nanu Alan Kachari, Mr. Nava Kumar Boro, and Mr. Bhriguraj Borah for resolving technical and engineering-related issues. Special thanks to Ms. Gauri Khuttiya Deori, Mr. Gourish Mazumder, Mr. Manojit Bhattacharjee, and Mr. Prabin Bharali for helping with all the administrative work. I would also like to thank all the fac-

ulty members and the staff for their constant help and support. I am also grateful to the security personnel for their support.

This endeavor of a Ph.D. would not have been possible without the financial support and resources from the esteemed Ministry of Education and my institute, IIT Guwahati. My sincere gratitude to the Computer and Communication Centre and the Lakshminath Bezbaroa Central Library for providing access to the resources throughout my Ph.D., especially during the difficult times of COVID by providing remote access.

Many thanks to my senior, Dr. Tushar Semwal, for his guidance during the initial phases of my Doctoral Research. He has immensely inspired me to keep working towards and deliver high-quality research work. I thank Dr. Priya Nair for her words of encouragement and motivation. I would also like to mention my seniors, Dr. Anasua Mitra, Dr. Pallabi Saikia, Dr. Aparajita Dutta, Dr. Madhurima Buragohain, and Dr. Deepankar Nankani for their guidance and support. Special thanks to Gayathri Rangu for helping me out with administrative work and for being an awesome mentee. I am incredibly grateful to Menaxi J Bagchi, Ajitem Joshi, Akul Agrawal, Shubham Sharma, Vinod Kumar, and Subhash for the discussions, late-night code and robot debugging sessions, and idea brainstorming, which made the Robotics Lab. a very happening place. I also thank my colleagues Vanshali Sharma, Arvind Agarwal, Bidyanath Jha, Suraj Pandey, and Sonia.

I am fortunate to have had Sanjukta Dutta, Nivedhitha S, and Jyoti Kainthola, who have stood by me and supported me throughout. Thanks should also go to my friends Fouziya Abbasali, Nikita Teggi, Aakansha Mishra, Aditya Hassan, Jeevitha, Pawan Mishra, Arijit Nath, Akshay Parekh, Jayprakash Nair, Nama Tako, Aanisha Akhtar, Manjari Saha, Meenu Dey, Priyanka Panigrahi and everyone who has been there for me. Last but not least, the dogs and cats of our campus have brought joy and happiness to me, making this journey relaxing and fun.

September 26, 2022 Divya D. Kulkarni

Declaration

I certify that

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and providing their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis is plagiarised to the best of my knowledge and understanding and that I will take complete responsibility of the same, if any complaint arises.
- I am fully aware that my thesis supervisor is not in a position to check for any possible instance of plagiarism within this submitted work.

September 26, 2022 Divya D. Kulkarni

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Dr. Shivashankar B. Nair
Professor
Email : sbnair@iitg.ac.in
Phone : +91-361-258-2356

Certificate

This is to certify that this thesis entitled "**Immuno-Inspired Embodied Lifelong Learning in Robots**" submitted by **Divya D. Kulkarni**, in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy, to the Indian Institute of Technology Guwahati, Assam, India, is a record of the bonafide research work carried out by her under my guidance and supervision at the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam, India. To the best of my knowledge, no part of the work reported in this thesis has been presented for the award of any degree at any other institution.

Date: September 26, 2022
Place: Guwahati

Prof. Shivashankar B. Nair
(Thesis Supervisor)

Abstract

Embodied artificial intelligence involves agents that learn by interacting with the environment directly. It fairs well and is robust compared to learning from curated datasets. Such datasets are constrained and do not cover varied conditions encountered in the direct interaction with the environment. Incorporating embodied learning with lifelong learning, where the learning adapts to changing environment and accumulates learned information over time, leads to a marked improvement in the whole learning process.

The biological processes in nature have inspired the formulation of algorithms to solve complex problems in the computational world. Evolutionary algorithms are one such class of algorithms inspired by biological evolution. They are perfect for embodied learning since it involves interacting and evolving as per the direct interactions with the environment. On the other hand, Artificial immune systems, inspired by the biological immune system, are ideal for achieving lifelong learning in the computational world as they can preserve the learned information, which can be reused as intended later. This thesis focuses on embodied lifelong learning to evolve robot controllers with evolutionary algorithms and artificial immune systems based methodologies.

The first contribution of the thesis focuses on embodied lifelong learning of robot controllers. Evolutionary robotics employs unconventional techniques to evolve controllers for robots based on their fitness values continuously. In most cases, a parent controller is subjected to mutation to evolve its offspring. If the offspring performs better than the parent, the former is made to replace the latter. This essentially results in a significant loss of information learned by the parents over generations. A straightforward workaround to circumvent this problem is to maintain a *Hall of Fame* (*HoF*) comprising the best parent controllers for use in future generations. In embodied evolutionary robotic scenarios, caching a large number of controllers in an *HoF* would result in increased computational overheads while

selecting the best out of them, resulting in a drastic reduction in performance. With no means to find an upper limit to the number of controllers that can populate an *HoF a priori*, devising a technique to dynamically regulate this population is imperative. In this contribution, a novel method to evict the non-performing controllers within an *HoF* based on the dynamics of the system is proposed. Experiments performed using simulations, and also a real robot indicate a marked improvement in the learning process due to the dynamic eviction policy.

The second contribution deals with enhancing the evolution of a single controller. Artificial Neural Networks (ANNs) are used as robot controllers. Apart from backpropagation, the ANNs are also evolved by neuroevolutionary techniques, where their weights are randomly mutated, after which the performance of the ANN is calculated. This contribution describes a novel mutation strategy that enhances the performance of random selection by augmenting a dedicated *mutational puissance* to every weight within the ANN. These *puissances* are either replenished or consumed based on whether the performance of the ANN improved or deteriorated. Mutational *puissance* thus, tends to preserve, in some form, the information on how well the associated weight contributed to the performance of the ANN. This technique in a robot enables it to evolve ANN-based controllers for specific tasks on-the-fly. The experiments and results portrayed in this conclusively endorse the efficacy of guiding mutation using the concept of *puissance* thereby enhancing the evolution of the ANN-based robot controllers.

The third contribution is neuronal transfer learning, where the information learned in one domain or environment is transferred to another to save the number of computations and also to accelerate learning in a new domain. Such transfer learning in Artificial Neural Networks (ANN) is performed by transferring some or all the layers of neurons from within an already learned source ANN to a target ANN which is then made to learn a new domain. Though such transfers are known to accelerate the convergence of the target ANN, there is no hard and fast method to ascertain which of the neurons or layers need to be transferred. While most methods advocate the transfer of complete layers of neurons, this contribution describes an Immuno-inspired transfer learning paradigm that aids in ascertaining the pertinent or *Hot* neurons, which, when transferred, facilitate faster convergence of a target ANN.

The method uses the concept of an *Idiotypic Network* to evolve the *temperatures* associated with neurons within the various layers of the source ANN. *Temperatures* of those neurons which contribute more to the learning, increase making them *hotter*. This paper also presents results obtained from experiments performed on dissimilar datasets using this method, which clearly authenticates its efficacy.

The fourth and final contribution aims to integrate transfer learning with the lifelong learning paradigm. The information learned by the controllers from the robot inhabiting an environment is transferred to another robot in a different environment. The target environment can be either similar or dissimilar compared to the environment of the source robot. In the first contribution, the *HoF* pertained to a single robot in an environment, whereas this contribution aims at transferring the *HoFs* between the robots of different environments. The results from the experiments indicate the acceleration in the learning and the efficacy of the neuronal transfer of controllers.

Contents

List of Figures

List of Algorithms

List of Tables

List of Symbols

Symbol	Description
ϵ	Boundary of Search Space or Cross Reactivity Threshold
Ab	Antibody
Ag	Antigen
Ep	Epitope
Pt	Paratope
∇	Set of Candidate Antibodies
χ	An Active Region
$\S_{Ag}$	Stimulation from an Antigen
S_{∇}	Stimulations or Suppressions received by a set of Candidate antibodies
F	Fitness value associated with an Antibody
C	Concentration value associated with an Antibody
Ctr	Controller associated with an Antibody
R	Resource associated with an Antibody
R_{max}	Maximum value of Resource
R_{rep}	Resource Replenished
R_{rep}	Resource Consumed
η	Resource Decay constant
ID	Unique identifier of an Antibody within the repertoire of Antibodies
HoF_{χ}	Hall of Fame of Antibodies of an active region χ
σ	Extent of mutation
σ_{max}	Maximum extent of mutation

λ	Moving average difference in the fitness values of Champ and the evolving mutants
ψ	*Mutational Puissance* value associated with a weight in an Artificial Neural Network
ψ_c	Amount of *Mutational Puissance* consumed
ψ_r	Amount of *Mutational Puissance* replenished
ψ_{max}	Maximum value of *Mutational Puissance*
μ_p	Moving average difference in the fitness of the parent of the previous generation and the current generation
μ_{ch}	Moving average difference in the fitness of the parent and the fitness of the child
κ	Decay constant of *Mutational Puissance*
θ	Temperature value associated with a neuron in an Artificial Neural Network
τ	Stimulation received by the neuron
ι	Suppression received by the neuron
ϑ	Top best neurons within a layer in Artificial Neural Network
δ	Number of neurons that are transferred
$\blacktriangleright^x$	Transfer of top x Hot Neurons from a layer
$\blacktriangleright^\lambda$	Transfer of the complete layer
E_S	Source Environment
E_T	Target Environment
Re_S	Source Repertoire
Re_T	Target Repertoire
Re_{Ex}	External Repertoire
p_{ex}	Random Probability of choosing from external Repertoire
$T_=$	Same tasks in E_S and E_T
$T_\neq$	Same tasks in E_S and E_T
$T_=^x$	Transferring top x hot neurons in $T_=$ case
$T_=^\lambda$	Transferring all neurons in a layer in $T_=$ case
$T_\neq^x$	Transferring top x hot neurons in $T_\neq$ case
$T_\neq^\lambda$	Transferring all neurons in a layer in $T_\neq$ case

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

1

Introduction

Embodied Artificial Intelligence (AI) involves algorithms where the agents learn by interacting with the environment directly. Learning is not performed using curated datasets but rather by direct interactions with the environment [32]. This type of learning can handle noisy and dynamic data in a given environment and is known to be robust. Agents, both soft and hard, can use embodied learning. While soft agents inhabit simulated worlds, robots which constitute their physical counterparts, interact with real worlds [15].

Lifelong learning is a process of learning where the learning is accumulated over time[101]. The learning process can thus, adapt to changing environmental conditions. This process also learns and adapts as and when the system encounters environmental conditions never seen earlier [55], making it different from embodied learning. Embodied lifelong learning refers to the learning process that fuses embodied and lifelong learning mechanisms. In this method, learning is performed within dynamic conditions. It can adapt to changing environments and is able to reuse the learned information in the past, while doing so.

1.1 Inspirations from Nature

Nature has been an efficient and optimal solutions provider for complex problems. This is true even in scenarios where there are complex interactions between het-

erogeneous entities. Biological processes observed in nature use simple yet effective strategies which have evolved over several thousands of years, thus making them robust and reliable. Several problems encountered in the domain of Computer Science and Engineering share a level of commonality with those in biological processes which seem to have been resolved eons ago in the biological world. Mimicking *Nature*'s strategies to solve problems in the computational world thus, seems reasonable [13, 38]. The translations of these biological processes encountered in *Nature* into the computational world are often termed *Bio-Inspired Algorithms*, and the underlying learning mechanisms fall under the category of *Bio-Inspired Machine Learning*.

Many such algorithms have been designed based on inspirations drawn from insects, animals, biological processes of living organisms, and even biological evolution. Insects being comparatively simpler organisms, have been a great source of inspiration. Though individual insects could seem dumb, they are known to exhibit exceptionally intelligent behaviors when grouped together in the form of swarms or colonies. Various types of ants, bees, and wasps have been studied, and associated algorithms have evolved from them. The Ant Colony Optimization (ACO) algorithm, modeled based on the pheromoning behavior exhibited by ants in large colonies [31] is one such example. The algorithm was inspired by the co-operative foraging behavior of ants in a colony.

Such behaviors from the animal kingdom have been a constant source of inspiration for solving problems. These may be either of an individualistic type or as a team or a flock. The Particle Swarm Optimization (PSO) algorithm which is one such, is based on the foraging behaviors exhibited by many avians. Introduced by Kennedy et al. [67], PSO works on a flock of birds whose computational equivalents are termed particles. Each particle changes its behavior based on - its internal feeling, that of its immediate neighbors (local), and that of the entire flock (global). These *feelings* are combined mathematically to produce the next position of the particle in the environment. There are numerous other classes of bio- or nature-inspired algorithms that model swarm intelligence. A review of such swarm-based algorithms can be found in [131, 23].

Based on the Darwinian evolutionary principles, where life forms were optimized and adapted, Evolutionary Algorithms (EA) [9] form one of the effective

methods for optimization in the computational world. Algorithms inspired by biological evolution have opened up a plethora of solutions in the area of machine learning [5].

Another class of bio-inspired algorithms is inspired by immunology. The immune system in vertebrates can effectively deal with trillions of antigens. Apart from a quick response, the system is highly adaptable, robust and also decentralized. Most importantly, the immune system remembers (memory) the manner in which the past antigens were eliminated [70]. Inspirations from the Biological Immune System (BIS) have resulted in a new paradigm known as the *Artificial Immune System* (AIS) which uses theoretical immunology to solve many a problem in the computational world [26, 120, 10].

One of the major application areas of bio-inspired computation is in the field of robotics. The vast diversity exhibited by processes in the biological world provides for classic inspirations to mimic them in the domain of robotics. Robots have been made to learn low and high level behaviors including locomotion, navigation, manipulation, swarm and co-operative behaviours, based on algorithms inspired from *Nature* [102].

1.2 Learning in Robots

A robot senses its environment through a set of on-board sensors and acts on the environment using its actuators. In the case of humanoid robots, the actuators could be pneumatic, hydraulic, or electric ones, whereas, for wheeled mobile robots, the motors attached to the wheels form the actuators. The controller of a robot decides which actions are to be taken by it. These controllers can be either pre-programmed to contain a set of *condition-action* rules or made to learn them dynamically on-the-fly. The former case is used when the environment and the rules of engagement with it are known *a priori* while in the latter case, a learning algorithm is integrated into the controller, which learns the actions to be performed as per the defined objectives. Based on how the controller performs within its environment, the robot receives *rewards* or *penalties*, which in turn enhance its learning. Learning of such controllers could be done offline with a human being in the loop. Once the learning process is

complete, the learned controller model is transferred to the robot. Learning could also be performed online and on-board the robot in an autonomous manner. Though online learning is more challenging and results in the actual use of the robot resulting in wear and tear, it is preferred due to the absence of the extra step of transferring the learned controller to the robot. In addition, learning online makes the robot tackle and adapt to changing environments more effectively.

The controller of a robot could comprise a set of simple $if-else$ rules or fuzzy logic-based rules [89]. Artificial Neural Networks (ANN) have also been widely used to control robots [40]. Due to their smooth input to output transitions and resistance to noise, ANNs are often preferred as robot controllers. In the work reported in this thesis, ANNs form the controllers of the robot.

Learning to control in autonomous robots is a non-trivial problem. Since conditions within the environment are always subject to change it is not feasible to handcraft every condition that a robot may encounter. The main challenges pertaining to the realization of robot controllers that can learn and adapt to changing environments are addressed below.

1.3 Research Challenges and Objectives

Realizing robot controllers that learn, entails numerous research challenges. Some of the more prominent methods used and the challenges they pose, have been described below.

1. Learning from Scratch (with no human intervention): The emerging paradigm shift to *Embodied AI* [32] makes use of intelligent algorithms and agents that no longer depend on datasets of images, videos, or text curated and made available *a priori* on the Internet or elsewhere. Agents that use Embodied AI learn by interacting directly with their environments using an egocentric perception, much like human beings. Embodied robots thus, have no knowledge of how to behave in their environments. Their controllers start learning and evolving from scratch with no human intervention by interacting and performing actions within their environment. While learning the controllers for a robot, it is ideal if the controller is not provided with any information beforehand as this may

at times lead to a bias and increase the number of computations to emerge out of this bias. Such a bias may hinder the learning of the robot's controllers or the learning may cease altogether, which is where the challenge lies.

2. Online and On-board Learning: As mentioned earlier, offline learning involves learning the controller without the real robot in picture and involves the additional step of deployment of the learned solutions on the real robot. Once deployed, such learning often cannot adapt to changes in the environment. Online-cum-On-board learning attempts to mitigate these shortcomings by making the robot learn by *sensorimotor* toil [17]. However, this type of learning has to cope with the computational resources and power constraints available on-board the robot without compromising on the quality and efficiency of learning, which poses a challenge.

3. Learning a complex task efficiently: When the robot is posed with a complex task in an online and on-board manner, learning to perform it is challenging due to power and computational constraints. Both time and energy in terms of power, could be wasted in the several trials that the robot performs during the learning process. There is also a possibility that the robot may eventually fail to learn such a complex task altogether. Addressing this issue without compromising the quality and efficiency of learning is, thus, vital.

4. Learning Controllers - *One or Many?*: Current AI approaches use Artificial Neural Networks (ANN), both shallow and deep, to learn aspects, using datasets, available *a priori*. Mimicking a similar style in the domain of robotics, an ANN-based controller could be made to learn behaviors a robot should possess in a given environment. Learning could be done offline if the data were available, or online, otherwise. However, real environments are dimensionally very complex and, worse, noisy. It is highly possible that an ANN-based controller would learn one task online and then later forget the same when another such task or behavior is to be learned. In brief, having just one controller may not solve the problem of learning several behaviours that need to be learned dynamically. This calls for multiple controllers, each of which can learn smaller sets of tasks or behaviors. This paradigm would,

however, result in the creation of several such controllers, handling of which can prove to be highly cumbersome and challenging, especially in resource-constrained robotic scenarios.

5. Preserving the Learned Controllers: When a robot has learned one aspect, it is essential to preserve the controller that has learnt it, so as to avoid redundancy in the learning exercise. Not doing so could lead to a wastage of both computational time and power. It is also necessary to identify which of these controllers need to be preserved, while also ensuring computational efficiency, which is a challenge.

6. Discarding Unused Controllers: Although it is necessary to preserve the learned controllers for future use, stashing many such controllers can make the robotic system sluggish. The fact that resources on-board the robot are generally sparse worsens the problem. Thus, it is essential to discard those that are comparatively ineffective and ones that are not being used by the robot. Deciding *when* and *which* of these controllers need to be discarded poses another challenge. Discarding a controller should be based on how effective and useful it has been all through the life of the robot.

7. Reuse of Learned Controllers: Controllers that have proved to be more effective in the past need to be preserved. However, reuse of the same by a robot, calls for a selection mechanism that is capable of selecting the best fit controller from these preserved ones. The selected controller needs to cater to the current state or condition of the robot. The selection mechanism thus, needs to be judicious.

8. Avoiding unnecessary computations: The evolution and hence learning of controllers should not be random. It should avoid unnecessary computations and focus on ensuring efficient learning and fast convergence.

9. Transfer of Learning: Having learnt to adapt and behave in an environment, it would be propitious if these learned controllers can be transferred to other robots struggling to learn the same or different behaviours/tasks in either similar or dissimilar environments. This would give a head-start for the robots

thereby decreasing on *sensorimotor* toil and consequently hastening learning convergence.

10. Deciding on What to Transfer: Transferring the whole learned controller to another robot(s) may not always be advantageous. In fact, often times, such transfers tend to worsen the learning exercise. This is true, especially when the source (where it was learnt) and target (where it is going to be used) environments are different. Such a transfer can, at times, decelerate and make the learning process go haywire. It is thus, vital to have an expedient method for selecting and transferring portions of the controllers that will ensure that the learning at the target side is enhanced.

This thesis aims to address these challenges discussed above. Proposing solutions to these challenges form the major objectives of the work reported in this thesis. The initial portions of the thesis focus on mechanisms which address the challenges in evolving robot controllers that learn from scratch in an online and on-board manner. Mechanisms to cache and reuse the more effective controllers and also discard the ineffective ones, have been covered. The thesis then concentrates on a single controller and describes techniques to make it learn efficiently by avoiding unnecessary computations, thereby leading to faster convergence. The latter part of the thesis delves into the novel mechanisms to transfer the right portions of the learned controller. It describes a method of finding these right portions which when transferred prove to be more effective than conventional transfers. Such a transfer of learning has been propounded and initially proven in the context of Deep Neural Networks (DNN), for both similar and dissimilar source and target datasets. Later the thesis reports of such a transfer effectively implemented among robots in both similar or dissimilar environments. The thesis thus, presents methods for both evolving efficient learning controllers from scratch and eventually transferring the pertinent portions of the learned models for use in other environments.

1.4 Background

This section presents a brief review of Evolutionary Algorithms and Artificial Immune Systems, both of which form the core inspirations in the development of the

mechanisms described in this thesis.

1.4.1 Evolutionary Algorithms

Evolutionary Algorithms (EA) form a class of bio-inspired algorithms that mimic the theory of biological evolution [6, 33]. They are based on the Darwinian theory of biological evolution, which explains the adaptive changes of the species by the principle of *Natural Selection* [7]. Natural selection favors those species that adapt to their environmental conditions for survival and further evolution. Differential Evolution [24], Genetic Algorithms (GA) [51, 57] and Genetic Programming (GP) [75] are few of the flavors of Evolutionary Computational algorithms.

Evolutionary algorithms model the collective learning process within a population of individuals (or genomes). In EA, a linear chain of integers represents an individual (or a genome). Initially, an algorithm-dependent method initializes the population of individuals. The population evolves towards the better regions of the search space through a process that includes selection, recombination and mutation. Crossover is a type of recombination where a certain portion of the two individuals are exchanged to create a new individual [116]. In the first phase of the evolutionary algorithm, the fitness of the individual is evaluated. The fitness value determines the quality of the new individuals in the space and is calculated based on a predefined *Objective* (or Fitness) *function*. After this *evaluation* phase, the individuals with higher fitness values, i.e., the qualitatively better ones, are often favored over those with lower quality in the selection process. This is followed by the *variation* phase, where either a crossover or mutation process is applied to the selected individuals. In the crossover, which is performed based on a certain probability, the parental information of two individuals is made to cross over to form two new descendants. The mutation process, which is carried out with a lower probability than that of crossover, introduces innovation into the population of individuals. Each cycle of this evaluation-selection-variation is termed as a *generation*. Generations continue till a stopping criterion is met, which could usually be a preset maximum number of generations or when an individual has achieved maximum fitness.

Variants of EA

Neuroevolution: In order to learn the weights of an Artificial Neural Network (ANN), backpropagation is the most preferred method. Evolutionary algorithms provide an excellent alternative to learning the weights of an ANN. The ANN is represented as a genome, and the mutational process is applied to tweak its weights [130]. Like the conventional EA, the fitness score decides the goodness of a mutated ANN. Some neuroevolutionary methods also tweak and evolve the architecture of the ANN in addition to its weights [39].

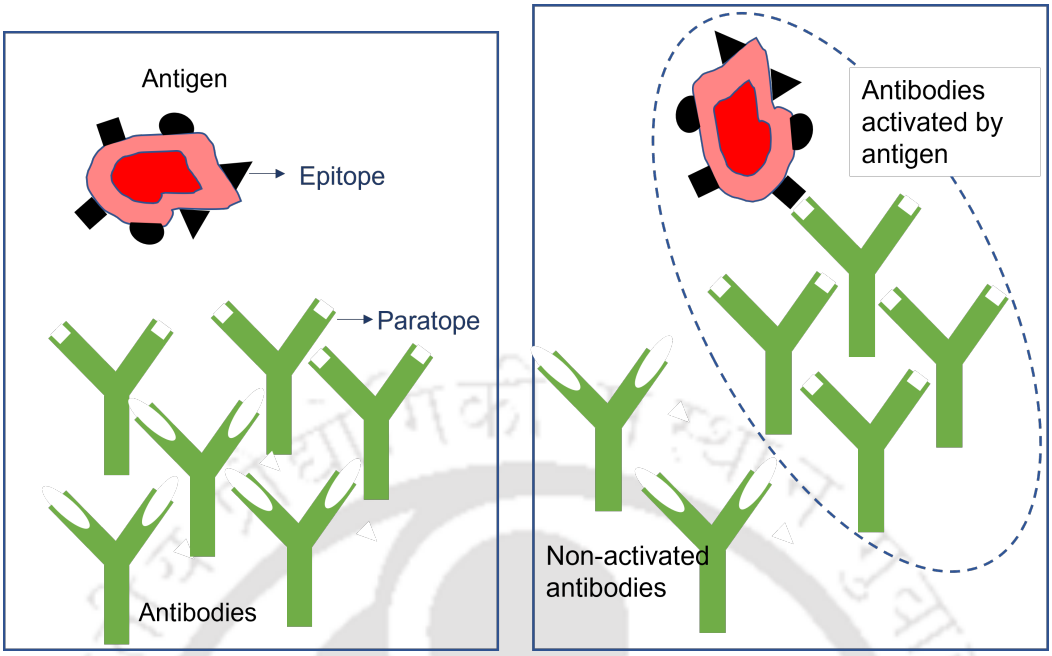
Evolutionary Strategies These variants of EA avoid recombination and make use of only selection and mutation [86, 42]. One such popular evolution strategy is the (1+1) Evolutionary strategy [14], where a single offspring is created by applying a mutation operator on a single individual. If the mutated offspring performs better (i.e. has a higher fitness) than its parent, then the former replaces the latter; else, the parent is retained and the extent of mutation performed on it, is increased, and the process continues. The parent genome is referred to as a *Champ* while the offsprings that are evolved from it, are called as *Challengers*.

Evolutionary Robotics

Evolutionary robotics applies the evaluation, selection and variation for the evolving controllers of the robots based on their fitness values. [30, 98]. Fitness functions depend on the desired behaviors or tasks that the robots are supposed to learn. Robots are made to interact with the environment using the evolved controllers, and the behaviors, which are being evolved, are observed. Based on the observed behaviors of the robot and the desired objectives, a predefined fitness function is used to rate the controllers. The individual controllers for the next generation are selected based on their fitness values. These selected ones are made to produce offsprings using variation techniques such as mutation and crossover.

Embodied Robotics

Embodied Evolution in Robotics refers to evolving the controllers of the robots onboard one or many robots [37]. Instead of learning from the datasets, offline and



(a) (b)

Figure 1.1: a) Depiction of antibodies and antigen with their respective paratopes and epitope b) Activated antibodies which recognize antigen and non-activated antibodies

offboard the robot, one or more robots learn by directly interacting with the environment [32]. This environment, either simulated or real world [108], can enforce different kinds of constraints on the robot. Since the environment is partially observable and not static, the evolution of robot controllers under such environments is always a challenge.

The evolutionary algorithms form an excellent choice for evolving controllers for robots. However, since they do not have a memory component associated with them, such controllers are prone to loss of learned information. In order to fully utilize the potential of an EA, overcoming this drawback is *sine quo non*.

1.4.2 Artificial Immune Systems

Artificial Immune Systems (AIS) are those bio-inspired algorithms that derive their computational intelligence by mimicking the complex processes that constitute a Biological Immune System (BIS).

The Biological Immune System (BIS) comprises a set of defense mechanisms

found only in vertebrates. Several biological processes protect the host against a large spectrum of invading antigens [25]. The primary aim of the immune system is to identify the *self* (host cells) and *non-self* cells (antigens) in the body and eliminate the latter. In order to do so, the immune system produces a range of cells such as B-cells, Cytotoxic T-cells, Helper T-cells, etc. These cells, in turn, secrete proteins known as antibodies. The primary function of these antibodies is to identify the antigens, tag and kill them and eventually aid their removal from the body. The recognition of the antigen by the antibody is based on the binding between the two. The extent of this binding is called *affinity*. More the binding, the better is the affinity. The binding site on the antibody is called the *paratope* and the complementary area on the antigen is called the *epitope*. While a specific antibody has the same type of paratopes, an antigen can have differently shaped epitopes (as shown in Fig. 1.1a). Antibodies that recognize an antigen (that is, when the paratopes are complementary to an epitope of the antigen) get activated. This behavior is illustrated in the figure 1.1b.

Various immune theories have been proposed in the domain of AIS, the prominent ones being Clonal selection, Danger theory, and Immune Network theory. The Clonal selection theory [2] states that whenever any antibody recognizes an antigen, a clonal expansion occurs, leading to an increase in its population. Somatic hypermutation [69] increases the average affinities of these clones towards the antigen that triggered the cloning. Clonal expansion also leads to the creation of *memory* cells, [125] which hold the information of the antigen to utilize them in the future, in the case of a similar attack. Danger theory [84] postulates that when a cell dies an unnatural death, it sends out danger signals and establishes a danger zone around itself [4]. When the antibodies sense these signals, they are attracted towards this zone where they capture and kill the antigens. Those having more affinity undergo clonal expansion. This increase in the population of antibodies that are better suited to the situation aids in quickly curtailing the growth of the antigens. The Immune network theory [61], on the other hand, propounds that antibodies on their own form a network even in the absence of an antigen. This network is called the Immune network. This network is formed based on stimulations and suppressions amongst the antibodies and their respective concentrations. Farmer et al. [36] designed the

equations which model these interactions between the antibodies and the invading antigen.

AIS-based learning exhibits the *Lifelong learning* phenomenon, a mechanism that not only improves the learning over time but also adapts and learns to adapt to the new conditions while also remembering how the antigens were effectively tackled. The antibodies of a BIS can learn to eliminate new invading antigens, remember the mode of tackling so that, in future if the same attack occurs, it can be quickly quelled. Adaptation to contain new antigenic attacks and a memory for each such attack, makes the BIS a perfect inspiration for achieving lifelong learning.

From a computational perspective, a population of antibodies is equivalent to a population of solutions in an EA. The respective affinities of the antibodies are synonymous to the fitness values of the solutions in an EA. It may be noted that, unlike in an EA, the antibodies can clone based on their affinities towards the antigens, thereby creating sub-populations of heterogeneous antibodies. An extensive survey of AIS has been reported by Dasgupta et al. [29, 27, 28]. AIS based algorithms have been applied in various areas of Computer Science such as computer security [44, 43, 68], data mining [52], optimization [53, 121], anomaly detection [48], pattern recognition [79], etc. AIS has also been widely used in the field of robotics [60, 114, 129, 94, 105, 104]. Hart, et al. have [56] reported that the majority of publications on AIS have been on benchmark problem instances rather than real-world problems. The authors have also reported that AIS-based applications in robotics are mostly *simulated* in artificial environments. The need to envisage and implement AIS based applications in physical environments is thus, imperative.

Combining EA and AIS for Learning Robot Controllers

The nature of the EA, the evaluation-selection-variation process, ensures that every condition encountered by the robot and the robot's corresponding action is taken into account and evaluated so as to aid the learning of the robot controllers. This thesis focuses on EA-based learning of robot controllers. However, as explained earlier, the EA is prone to loss of learned information in the past, during the evolutionary process. On the other hand, an AIS is a much more promising technique to

adapt to dynamic conditions without the loss of such learned information. Such a feature, if incorporated within a robot's controller, will drastically ameliorate its performance especially when the environmental conditions are complex and dynamic. This work reported in this thesis, thus describes such an integration of AIS and EA based techniques, that culminates in an apt recipe for *lifelong learning* in the domain of robotics.

1.5 Contributions of the Thesis

This section describes the significant contributions of the thesis in brief. The first contribution involves learning the controllers from scratch, preserving and reusing (or caching) them efficiently to realize embodied lifelong learning. The next contribution focuses on enhancing the evolution of the individual controllers. This is followed by a novel mechanism for neuronal level transfer learning in Deep Neural Networks. The last contribution integrates lifelong learning with this mechanism for transfer learning and embeds it in robot controllers to enable their transfer from one environment to another.

1.6 Embodied Lifelong learning

This contribution highlights embodied learning from scratch and aims to evolve controllers for a robot in an online and on-board manner, starting with no previous information. At the core of each of the robot's controllers, is an ANN. It may be noted that one robot has several such ANNs each constituting a controller. The robot's workspace is defined by the set of all vectors each comprising the values obtained from its sensors at a particular instant of time. A subset of these vectors which are close-by (within a defined radius) constitutes an *active region*. Controllers are evolved for each such active region.

The initial parent controller gets replaced by its mutated offspring if the latter outperforms the former. The eviction of the parent controller also leads to the loss of learning achieved by the parent. In this contribution, we propose a novel method to maintain a *Hall of Fame (HoF)* of controllers comprising the best parents, which can be reused in future generations. Instead of evicting the parent controller upon being

replaced by the offspring, it is cached in an *HoF*. However, caching a large number of controllers can lead to an increase in computation, thereby drastically reducing performance. One way out is to limit the number of controllers cached in the *HoF*. There is however, no real method to establish the optimal value of the number of controllers to be cached especially when the learning is carried out in an online and on-board manner. It is thus, essential that the system adapt to the changing environment and regulate the number of controllers it needs to store in an *HoF* by adding or evicting the cached ones as and when required. This dynamic regulation of the number of controllers residing in an *HoF* calls for a proper mechanism for eviction of the non-performing ones based on the current state of the system. The formulation of this mechanism forms the essence of this contribution. Using two new concepts - *Resource* and *Concentration*, a mechanism to determine which of the controllers need to be re-selected or evicted from the *HoF* has been described.

This mechanism features:

1. Caching of the parent controller in a *HoF* upon being replaced by the offspring controller

2. Evicting controllers based on the dynamics of the system

3. Can cater to both simulated and real-world robots

Experiments were performed both on real and simulated robots with and without the use of the concept of *HoF*. In the *non-HoF* based work, there was no caching of controllers, dynamic regulation or eviction. Experimental results show a remarkable performance in the learning when *HoF* based method is used as compared to the one which does not use it. The results also show the importance of the use of *Resource* and *Concentration* in the eviction and re-selection of the controllers from an *HoF*, respectively. This contribution helped achieve the evolution of high-performing controllers from scratch, under various conditions, even for a complex task viz. *Phototaxis cum Obstacle Avoidance*, while dynamically maintaining the required number of controllers in *HoF*.

1.7 Enchancing Neuroevolution

This contribution focuses on ameliorating the evolution of a single controller in robots. The controller, which drives the robot, is an *Artificial Neural Network (ANN)*. During the evolution of these ANN-based controllers, an offspring controller is obtained by randomly mutating all the weights of the parent controller. For such tweaking of weights, a Gaussian-based mutation process is followed. However, if all the weights are randomly mutated simultaneously, the performance may degrade. Such performance degradation is observed because weights that have attained their optimal values also get disturbed during the mutations. In this work, we propose a novel method for mutation of weights by augmenting a new concept termed *mutational puissance*, to every weight in the ANN. This *puissance* value, associated with every weight in the ANN, is increased or decreased based on whether the performance of the ANN improves or degrades. Thus, *puissance* indicates how each of the weights has individually contributed to the performance of the ANN over time. Such a *puissance* guided mutational strategy tends to accelerates the evolution of the ANN, thereby improving its performance over time. Experiments were conducted to evolve ANN-based controllers for the robot using this method. Results obtained clearly indicated that the *puissance* based method works better than the methods where all the weights are mutated in a random fashion.

1.8 Neuronal Transfer Learning

If the information learned in one domain or environment could be transferred to another, it can save a large amount of time and computation and also accelerate the learning process in the new domain/environment. This is true, even in the world of robotics.

Such a transfer can also aid the reusability of learning, not just within the system but also across systems. This contribution comprises a novel mechanism to transfer the learning achieved from one ANN to another. The ANN from which the learning is being transferred is termed the *source* ANN, and the one where it is transferred is termed, the *target* ANN. Conventional transfer learning methods transfer a whole layer from the source ANN to the target ANN. Such layer-based

transfers are known to be beneficial only when the source and target ANNs deal with similar datasets to learn. In other words, such a transfer works when the source and target domains are alike. Transfer learning for dissimilar domains has hardly been successful. This work introduces a novel neuronal level transfer from the source to the target ANN, which can work in cases where the source and target datasets are dissimilar.

The results obtained from experiments on both shallow and deep neural networks establish the efficacy of the proposed method. The results indicate that the proposed method works best in the transfer of learning in the ANNs with learning similar data. They also highlight remarkable improvements in the case of dissimilar data. Every transferred layer in the source DNN was divided into populations, and the top 2, 3, and 4 neurons were transferred from each population to the respective target layers. It was observed that the lower order transfer of neurons leads to better accuracies in the target DNNs. This work proves the effectiveness of the neuronal level transfer in both shallow and deep ANNs. In the following contribution, we focus on using this method to transfer the learned information in the robotics domain.

1.9 Integrating Lifelong learning and Transfer Learning

This contribution focuses on transferring the information learned by the controllers from the robot inhabiting an environment to another robot in a different environment. The environment of the target robot can be either similar or dissimilar compared to the source robot's environment. In the work described in 1.6, the Hall of Fame (HoF) pertains to a single robot in its environment. In contrast, the work presented in this thesis introduces a strategy to transfer the HoF between the robots of different environments, i.e., from the source robot to the target robot residing in a similar or dissimilar environment. Instead of the conventional method of transferring the complete HoF from the source to the target, we focus on selecting and transferring only pertinent neurons from the controllers within the source HoF to those of the target.

The results from the experiments bring out the efficacy of such a neuronal

transfer of controllers from one robot to another residing in a different environment. It also accelerates the learning of the controllers in the target robot.

1.10 Outline of the Thesis

The thesis comprises six chapters. The chapter wise organization of the thesis is given below:

1. Chapter 1: This chapter provides an introduction, survey, and the motivation for the research presented. The major terminologies that appear in the thesis are also discussed.

2. Chapter 2: The work done in evolving the controllers for the robot from scratch - including preserving, reusing (i.e., caching), and evicting the controllers are covered in this chapter together with the results obtained. The contents of this chapter have been published in [74].

3. Chapter 3: The Chapter elaborates on the efficient neuroevolution of a robot's ANN based controller. Experimental results are also included herein. The contents of this chapter are based on the work published in [72].

4. Chapter 4: In this chapter, the method of neuronal level transfer from a source ANN to the target ANN has been described, together with the results obtained from experiments performed. The chapter is based on the work published in [73].

5. Chapter 5: This chapter describes how the work reported in previous chapters can be implemented in robots. Results obtained from such an implementation have been presented to emphasize that the contributions of the thesis can be used in the domain of robotics.

6. Chapter 6: The conclusions and future research directions are presented in this final chapter.

2

Embodied Lifelong Learning

Evolutionary Robotics (ER) deals with the process of evolving controllers for robots that mimic natural evolution, which consequently aids in learning to execute a set of given tasks. Initially, a set of chromosomes, which constitute the initial population, are randomly generated. These chromosomes are decoded into controllers, downloaded onto a robot which in turn executes the task(s) in accordance with the controller. While a robot carries out its tasks in an environment, the performance of its controller is evaluated continuously.

These chromosomes or controllers are made to evolve into newer ones using genetic operators such as reproduction, mutation, and crossover, after which the individuals of this newly generated population are evaluated. This evolutionary process is continued until the conditions (such as the performance score, number of generations, etc.) set by the user are eventually satisfied [97]. Such robot controllers are either evolved offline and then transferred onto the real robot, or directly evolved on-board the robot during run time (online). In the on-board evolution of controllers, the evolution is autonomous and occurs while the robot actually executes the given task in the environment.

Most often each controller constitutes either a set of *if-else* rules, a Proportional–Integral–Derivative (PID) controller or an Artificial Neural Network (ANN). Due to their robustness to noise and a smooth input to output transformation, ANNs have proved to be a more popular choice while modelling robot controllers

[11, 41, 113]. Controllers are evolved to solve a defined set of tasks and are evaluated using a fitness function [92]. The fitness function exerts a selection pressure on the evolutionary process such that controllers with better fitness values evolve.

Traditional ER methods make use of a single controller that is evolved to accomplish a given task. However, if the task is complex, the evolutionary process suffers from bootstrap and deception issues [112]. Bootstrap issues occur when the fitness function fails to apply substantial selection pressure to evolve relevant controllers with higher fitness values. Deception issues arise when the controller tends to saturate at a local optimum and is deceived into a state that makes it cease further learning. Incremental evolution, human-in-the-loop, or behavioral decomposition [112] can be used to avoid such issues. In incremental evolution, a complex task is aimed to evolve incrementally by decomposing it into individual easily solvable components. The experimenter manually switches the evolution from one component to another. The human-in-the-loop approach involves a human supervisor to guide the search away from the local optimum. Behavioral decomposition [20] is an approach, where the task to be learned by a robot is divided into various subtasks, and the corresponding controllers are evolved to solve them. An arbitrator then performs the job of selecting the right controller that could be effectively used to execute a given subtask.

Semwal et al. [110], introduce an immunology inspired distributed, embodied action-evolution cum selection algorithm, wherein different controllers for different *regions* of the search space are evolved. The algorithm evolves and selects the appropriate controller, which has the highest fitness value corresponding to the antigenic space. The boundary of the search space for the evolution of a single controller associated with a subtask is tuned using a hyper parameter called the cross-reactivity threshold, ϵ. In their work they have reported that their approach has outperformed the (1+1) Online Evolutionary Algorithm ((1+1) Online EA), in which a single generic controller is evolved for the whole of the search space.

Generally in the evolution of controllers, if the offspring controller outperforms its parent, the former replaces the latter. When the controllers are modelled using Artificial Neural Networks (ANN), the past historical performance of the parent ANN based controller is ignored and it is evicted from the system as and when it

shows a sudden drop in fitness. The eviction of the parent from the system, results in losing all the learned information acquired by the evicted parent controller so far. The currently selected offspring may have performed well in a space where the parent was poor in doing so. There is, however, no assurance that this offspring will perform well in other regions of the search space where the parent exhibited better performance. This strategy will thus, force the offspring to toil again and evolve to learn to cope up in spaces where the parent had already learned to perform well earlier. This phenomenon can be attributed to *catastrophic forgetting* [45] in ANNs. Such a reset in the learning of a task by the ANN based controller is of significant concern, where most of the aspects will need to be re-learned, thus resulting in unnecessary delays and computational overheads in the evolutionary learning process.

One possible workaround to circumvent this inherent forgetting is to cache the best controllers evolved in the past and re-use them as and when required. The cache which is much like a *Hall-of-Fame (HoF)* [107], comprises the best controllers encountered so far. Whenever the fitness of the current controller drops below that of any in the *HoF*, the best controller from within the *HoF* is selected as the next contender.

This chapter describes an enhanced version of an earlier work on an immuno-inspired action-evolution cum selection algorithm reported in [110], referred to herein as *iAES*, by augmenting it with *Halls-of-Fame* that individually caches and evicts the controllers within, based on the dynamics of the system. The proposed *iAES-HoF* algorithm follows an embodied approach, where the evolution of controllers is the result of the algorithm running on a single robot. The dynamics of the system described in this work controls the number of controllers to be maintained in an *HoF* and also decides which of them needs to be retained or evicted. The system dynamics also decide the re-selection of a controller from the *HoF*. At first sight, one may strongly feel that the the eviction and re-selection mechanism could use Quality-Diversity (QD) based algorithms [103]. However, retaining a diverse set of controllers based on QD algorithms, may result in the eviction of controllers that have been exposed to various conditions. Such controllers inherently possess better experience and more knowledge in terms of tackling a variety of inputs. The basic

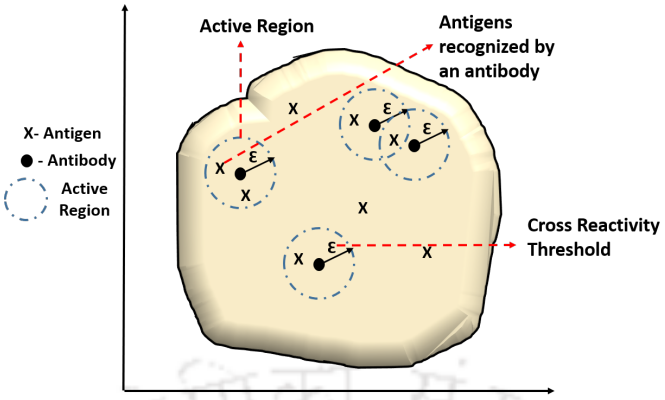


Figure 2.1: A Shape Space depicting the Active Regions within

objective here, is to retain controllers which are experienced and have exhibited better performance over time rather than diverse ones. QD based algorithms would tend to retain, diverse but inexperienced controllers, which is the reason why use of such algorithms has been avoided.

This work introduces the concept of a *Resource* associated with a controller, similar to the one referred by [50]. The *Resource* is either consumed or replenished based on how well the associated controller performs. It inherently carries information about the past performance of the controller and can thus be used to decide the retention or eviction of the associated controller from the *HoF*. The concept of the *Concentration* for the re-selection of the controllers from the *HoF* is also introduced. This work presents results based on experiments performed in simulation and also using a real robot. The effects of variations in ϵ, which alter the extent of search space, and the effects of *Concentration* and the *Resource* of the controllers in the re-selection and eviction of the controllers from the *HoF*, have also been studied and presented.

2.1 Biological Immune System

The primary aim of the Biological Immune System (BIS) is to identify the self and non-self (antigen) entities in the body and to eliminate the latter. In order to do so, the immune system produces various cells, such as B-cells, Cytotoxic T-cells and Helper T-cells. These cells, in turn, secrete proteins known as *antibodies*. The primary function of an antibody is to identify the antigens and tag them to

be eventually removed from the body. Recognition is based on the binding of the antibody (Ab) on the immune cell with the antigen (Ag). This binding site on the antibody is called the *Paratope (Pt)*. The complementary area on the antigen Ag, is referred to as the *Epitope (Ep)*. An immune cell has a large number of antibodies, all of which have the same type (shape) of *paratopes*. They are thus, said to be monoclonal. An antigen, however, can have differently shaped *epitopes*. The *affinity* (ξ) determines the extent of binding between the *Paratope* and the *Epitope*.

2.2 Artificial Immune System

An Artificial Immune System (AIS) is the counterpart of a BIS in the computational world. In an AIS, the state of the environment is represented as an Antigen (Ag) which is in the form of a vector. The Antibody (Ab) is that entity which takes an appropriate action to deal with the current environmental state, which is defined by the Ag. Antibodies which have high paratope to epitope affinities are the ones that are selected to quell the antigen(s). Since affinity need not be cent percent (i.e. an exact complementary match) any given antibody can recognize the antigens whose *Epitopes* lie in a part of a finite shape space as depicted in the Figure 2.1 . The small region, referred to as the *Active Region* (χ) of an antibody, is characterized by a parameter called the *Cross Reactivity Threshold* (ϵ). The smaller the value of ϵ, smaller is the active region, χ, which in turn means that it would require more distinct antibodies to cover a given shape space. On the contrary, a greater value of ϵ would mean that each χ covers a larger area, reducing the number of distinct active regions (χs) within the shape space. This also means a lesser number of distinct antibodies. It may be noted that an antigen Ag whose affinity is lesser than the ϵ of an antibody Ab will stimulate all those antibodies lying in the active region of Ab. Antibodies stimulated by an antigen, clone themselves, thereby increasing their concentration and consequently curbing the antigenic population. It may be possible that an antigen lies in more than one active region, as depicted in the overlapped regions Figure 2.1 . Since two antibodies are involved herein, the antigen is curbed by that antibody which has greater affinity with the antigen.

Immune Network Theory - According to the Immune Network theory or the Idiotypic Network [61], the antibodies form a network among themselves. Apart from the paratope, the antibodies also have an *Idiotope (Id)* that can match with the paratopes of other antibodies. An antibody is stimulated when its paratope matches complementarily with the epitope of an antigen or when its idiotope matches in a similar manner with the paratope of another antibody; else it results in its suppression. Stimulations and suppressions respectively, lead to an increase and decrease in the concentration of that antibody. The relevance of these metaphors to the proposed work described herein, is detailed in the section 2.4.

2.3 Related Work

This section provides a brief overview of how an AIS has been used in the realm of robotics. *HoF* based methodologies are also discussed.

2.3.1 AIS in Robotics

Artificial Immune Systems (AIS) have been used extensively to generate and arbitrate behaviors in robots.

In the Biological Immune System (BIS), a non-self entity or invader is termed as an *antigen* while an entity belonging to BIS responsible for eliminating these antigens is called as an *antibody*. In an AIS, often times the *learned behaviors* emulate the antibodies while the environmental state of the system, acts as the antigen.

Ishiguro et al. [60] state that the immune system provides a new paradigm suitable for dealing with dynamic problems with unknown environments rather static ones. They have proposed a BIS inspired approach to behavior based artificial intelligence. They have applied the proposed method to control the behavior of an autonomous mobile robot in a cluttered environment.

Lee and Sim [77] have proposed an immune system based cooperative control method for distributed autonomous robotic systems. They have modeled the environmental condition as an antigen and the behavioral strategy as the antibody. The robot selects the appropriate behavioral strategy based on the environmen-

tal condition which is then either suppressed or stimulated by other robots via a communication medium.

Watanabe et al. [124] present an immune-inspired method for decentralized behavior arbitration. This method has been applied to a garbage-collecting autonomous mobile robot. Adaptation mechanisms such as adjustment and innovation have been incorporated where the former relates to the change in system parameters and the latter corresponds to the adaptation in the behavioral selection process.

Singh and Nair [114] have proposed a mechanism for behavior arbitration combining the innate and adaptive characteristics of an immune system. Their robot learns to detect vulnerable areas of a track and adapts to the required speed while moving over these portions. Their experiments were carried out using two real robots deployed on two concentric tracks. The robot on the outer track assists the one on the inner track in case of misalignment of the latter. The misaligned robot records the unsafe conditions and learns to detect and adapt to such conditions in the future.

Whitbrook et al. [129] have integrated an Idiotypic Artificial Immune System (AIS) network with Reinforcement Learning (RL) based control system. They have used three systems : basic RL, a simplified hybrid AIS-RL and a full hybrid AIS-RL to test their hypotheses that account for the network advantage. They have tested the hypotheses on a test bed comprising a Pioneer robot which navigates through maze worlds detecting and tracking the door markers.

Nikhil et al. [94] have augmented the Idiotypic Network theory [62] with information sharing to make multiple robots learn and adapt their behaviors to changes in the environment. Using real robots they demonstrate how robots need to learn the correct behaviors to push colored objects into bins of matching colors. The robots learn to adapt and alter their behaviors even when the rules governing the environment are changed.

In the work proposed herein, an *Idiotypic Network* has been leveraged for the selection of the best controller from the *Hall of Fame* of controllers.

2.3.2 Hall-of-Fame Approach

The Hall-of-Fame (*HoF*) [107] approach has been widely used in conjunction with co-evolutionary techniques. The approach relies on the fact that some of the best performing individuals evolved over generations are cached and used as and when needed, in subsequent generations. In co-evolution, saving good performing individuals from prior generations drives each of the species to increase their respective levels of performance and complexity. Rosin and Blew [107] conducted experiments to play the games of Nim and 3-D Tic-Tac-Toe using a fixed the number of members in the *Hall of Fame*. The evolving individuals were pitched against the ones stored in the *HoF*.

Nicolai and Hilderman [93] evolved agents which can play the *no-limit texas hold'em* poker using co-evolutionary heuristics along with an *HoF* which had a limit on the number of agents. They have reported an improvement in performance when the *HoF* strategy was incorporated along with the co-evolution. The authors thus emphasize the need of using and maintaining an *HoF*.

Nogueira et al. [95] have analysed various *HoF* strategies in the application of competitive co-evolution for finding winning strategies in a two-player real time strategy, RoboWars game. They have tested five different policies for the deletion of the individuals stored in the *HoF* which either never delete the individuals or delete a varying percentage of the individuals in the *HoF*. This deletion percentage forms a hyper parameter and needs to be set by the user. They have shown that updating the *HoF* based on how diverse the individuals are, allows the system to exhibit the better performance. The diversity is measured by calculating the genotypic distance between every pair of the individuals in the *HoF*. Since this process has to be performed for every generation, it can be computationally expensive especially when the number of individuals is high. Their method, however requires the hyper parameters viz. the threshold distance and the percentage of individuals to be deleted, to be found empirically.

In the work described in [14], Bredeche et al. store the past ten best controllers in an *HoF* of controllers. These stored controllers were then validated and assessed in terms of their efficiency during the course of evolution. Here too the authors

emphasize the need to keep track of the best genomes and later reuse them for exploitation.

It has thus been noted that researchers who have used the concept of an *HoF*, have always fixed the number of individuals within, *a priori*. This essentially means that the maximum number of individuals that can reside in the *HoF* has to be somehow fixed, either empirically or otherwise, and then set as a hyper parameter. The eviction, the deletion percentages and diversity threshold also need to be set as hyper parameters. Subsequent sections describe a technique that can maintain individuals resident in an *HoF* based on the dynamics of the system.

2.4 Methodology

In the *iAES* algorithm [110], a given task was decomposed into several subtasks and separate ANN based controllers were evolved to perform each of them in a bottom-up manner. Based on the perceived sensory information (antigen), an ANN based controller (antibody) which is best suited was selected to perform the subtask and eventually evolve, if successful. This selection procedure involves comparing the distances between the epitope of the antigen and the paratopes of all the controllers (antibodies). The one with the maximum affinity i.e. the least distance, was selected to perform the subtask. The quality of the controller was measured by a user-defined fitness function after the subtask was executed. If the offspring derived by mutating the parent controller performed better than the parent, then the former replaced the latter. In the *iAES* algorithm, the search space is divided into *Active Regions* dynamically without human intervention. Each of these *Active Regions* is formed, and corresponding controllers are evolved on-the-fly. Further, the extent of each active region in the shape space is decided by the parameter ϵ, which is a hyper-parameter. Even when the search space expands, new active regions and associated controllers evolve dynamically. Hence, the *iAES* algorithm aids in real-time decision-making quite effectively. In their technique [110], they evict a parent the very first time itself, it performs badly. If the parent ANN based controller had been consistently performing well for several generations, it would most likely mean that it had gathered more information about the search space

than the offspring formed using mutation. In such a case, if this parent were to be evicted the very first time it fairs worse than the offspring, there would be a large loss of information learnt by it, from experiences in the past generations. Further, there is no guarantee that the offspring will perform as good as the parent in future generations. The information learned by the parent may need to be re-learnt by the offspring by sensorimotor toil, all of which culminates in redundancy and a waste of time and energy. Intuitively, one arrives at the conclusion that a controller should not necessarily be evicted the moment it performs worse than its offspring. Maintaining a cache or a *Hall of Fame (HoF)* for each subtask, comprising the best controllers over generations could circumvent this problem. With many good quality controllers evolved over the generations, it is highly likely that these caches would become over-populated very soon. The larger the number of generations, the larger would be the number of controllers in these caches consequently increasing the number of affinity comparisons to be made, thereby increasing the number of computations as well. In an embodied learning scenario, where learning is online and on-board the robot, such computational overheads will tend to increase response times thereby affecting the performance. One naive method would be to fix the upper limits of the number of controllers that could populate each of the caches. Since a cache is associated with every subtask, if there are T subtasks then there would be T such hyper parameters whose values would have to be empirically found and fixed *a priori*. However, the tasks are divided into subtasks online. Since the number of these subtasks is not known *a priori*, the number of associated caches also remains unknown. Thus, finding and fixing the upper limits of the number of controllers that can populate each of the caches, *a priori*, becomes a near impossible task. In the work described herein, a methodology is proposed to regulate the number of controllers in such caches or *Halls of Fame* based on the dynamics of the evolving system in an online manner. Two parameters viz. *Concentration* and *Resource*, have been defined for each controller which govern this regulation mechanism in a dedicated manner. While *Concentration* determines the re-selection of a controller from the *HoF*, *Resource* ascertains whether a controller needs to be purged from the associated *HoF*. Both re-selection and purging are based on whether a controller has performed better than the others in the associated *HoF*. The proposed method thus

regulates the number of controllers in the *HoFs* and endeavors to retain the better performing controllers. Results obtained from experiments performed using both simulations and a real robot show that the learning process improves significantly when such a *HoF* regulation mechanism was used.

In the further sections, the structures of the antibody and the antigen from the perspective of the proposed *iAES-HoF* algorithm are elaborated and also the inherent meta-dynamics used to maintain the *HoF* is described and finally, the *iAES-HoF* algorithm is presented.

2.4.1 Structure of Antigen and Antibody

The antigen Ag in the present context, is an N-dimensional vector where N is the number of sensors on-board the robot. The values of the sensors sampled from the environment constitute the values of the antigenic vector. The structures of the antigen and an antibody are illustrated in Figure 2.2a.

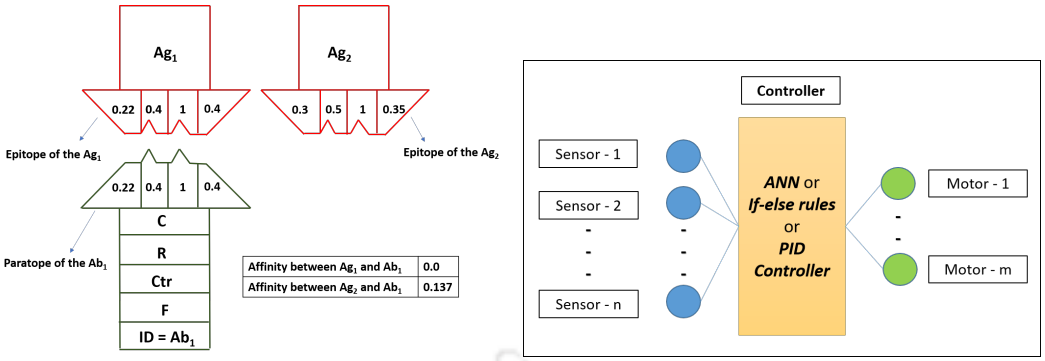
An antigen comprising an *Eptiope*, is matched with the *Paratope* of an antibody. As illustrated in the Figure 2.2a, the epitope is a vector comprising the values of the sensors on-board the robot.

An antibody Ab, shown in Figure 2.2a, in addition to a Paratope (Pt), consists of a Controller (Ctr), the Concentration (C) of Ab, the Fitness value (F) of Ctr, a *Resource* (R) associated with Ab and the *Identifier (ID)* of Ab, each of which has been described below.

Paratope (Pt) - When any antigen is encountered for the first time, the antigenic epitope and the paratopes of the existing antibodies in the *HoF* are matched. If there is a match, then the antibody is said to be able to deal with this antigen. If no match is found with any of the existing antibodies, then a new antibody is created and the epitope of the encountered antigen is set as the paratope of the newly created antibody.

Concentration (C) - Concentration is the cumulative effect of suppressions and stimulations received by a particular antibody from the antigen and other antibodies, since its inception in the *HoF*. The dynamics involved with the calculation of concentration is explained in detail in the section 2.4.2.

Resource (R) - The *Resource* of an antibody indicates how well it has performed



(a) (b)

Figure 2.2: (a) Structure of the *Ag* and *Ab* (b) A Controller (Ctr) connected to sensors and actuators(motors)

Biological Immune System	**Equivalents in the proposed Artificial Immune System**
Antigenic Epitope (Ep)	An N-dimensional vector obtained from a set of sensors on-board the robot
Antibody (Ab)	Controller based on an ANN
Paratope (Pt)	An N-dimensional vector which is compared with that obtained from the sensors
Affinity	Euclidian distance between the above two N-dimensional vectors

Table 2.1: Some immunological metaphors in the proposed work

since its inception in the *HoF* of antibodies and how soon the antibody is likely to be purged from the *HoF*. The calculation of the resource is detailed in the section 2.4.4.

Controller (Ctr) - A Controller can be a set of *if-else* rules, a PID controller or an Artificial Neural Network. Figure 2.2b shows a generic version of such a controller connected to sensors and actuators. In the proposed work, *Ctr* is a vector which comprises the weights of the Artificial Neural Network. The weights form the controller that drives the robot.

Fitness (F) - Fitness is obtained using the user provided fitness function for the given task. The fitness function is task-dependent. Fitness denotes how well the controller associated with an antibody is capable of performing a defined set of tasks. It is thus a measure of the quality of the associated controller.

Identifier (ID) - The identifier is a number which uniquely identifies an anti-

body in the repertoire of antibodies.

Figure 2.2a illustrates two antigens and the corresponding antibody when there is an obstacle in the front and to the left of a robot. The epitope of an antigen and the paratope of an antibody comprise the normalized values of the sensors attached to the robot. The other parameters stored within an antibody are also depicted in Figure 2.2a. Some of the pertinent immunological metaphors used in proposed work is summarized in the Table 2.1 for the sake of clarity.

The affinity (ξ) between an antigen and an antibody is calculated based on the Euclidean distance between the epitope of an antigen and the paratope of an antibody. Lesser the Euclidian distance, greater is the affinity. Only if ξ is lesser than or equal to the cross-reactivity threshold (ϵ), can an antibody tackle the antigen. In the Figure 2.2a, both the antigens Ag_1 and Ag_2 can be tackled by the same antibody Ab_1. This is because, the affinity between the antigens Ag_1 and Ag_2 and the Ab_1 is less than the value of the ϵ, when ϵ is assumed to be 0.3.

2.4.2 The Immune Network

According to the Immune Network theory [61], the antibodies form a network and suppress and stimulate one another. Whenever an antigen is detected in the environment, a set of antibodies, whose ξ is lesser than ϵ, gets stimulated. Since each antibody within this set becomes a possible candidate that can curb the antigenic attack, they are referred to as the set of *Candidate Antibodies*. The best one from among these *Candidate Antibodies*, is chosen to tackle the antigen. If the antigen is effectively tackled, all other antibodies in the set of *Candidate Antibodies* stimulate this best antibody. This best antibody, in turn, suppresses all other *Candidate Antibodies*. Stimulations result in an increase in the concentrations of the respective antibodies while suppressions decrease the same. This work uses a variant of Farmer's equation [35] to model the change in the concentrations of the antibodies.

During the antigenic attack, all the candidate antibodies receive a stimulation from the antigen which is given by the equation below:

$$S_{Ag} = \alpha * \xi \tag{2.1}$$

where S_{Ag} is the stimulation from an antigen Ag, α is a positive magnifying constant and ξ is the affinity between the respective antibodies and the antigen. The affinity between an antibody and an antigen is calculated as per the equation below:

$$\xi = EUC(Ep_{Ag}, Pt_{Ab_j}) \tag{2.2}$$

where ξ is the affinity between the antigen Ag and the antibody Ab_j, Ep is the epitope of the antigen Ag, Pt is the paratope of the antibody Ab_j and EUC is the Euclidean distance between the paratope and the epitope.

As mentioned earlier, the best antibody, with the highest fitness, out of the candidate antibodies receives stimulations from other candidate antibodies. All other candidate antibodies, apart from the best antibody receive suppressions from the best antibody. The stimulations or suppressions are calculated as per the equation below:

$$S_{\nabla_i} = \beta * \frac{C_{\nabla_i}}{\sum_{k \in \nabla} C_{\nabla_k}} \tag{2.3}$$

where ∇ is the set of candidate antibodies stimulated by the Ag at time t, S_{∇_i} are the stimulation/s or suppression/s received by the i^{th} antibody in the set ∇, β is a positive magnifying constant, and ∇_k is the k^{th} Ab in the set ∇. S_{∇_i} is used while calculating net stimulations and suppressions. Note that α and β merely magnify the the respectively associated terms and as such do not play any role in the dynamics.

Change in Concentrations: When the antibodies are stimulated or suppressed, their concentrations increase or decrease proportionately. The concentrations of all the antibodies in the candidate set increase due to stimulations from the antigen. In addition the concentration of the best antibody among the candidates will further increase due to the *stimulations* received by it from other candidate antibodies, as per the equation below:

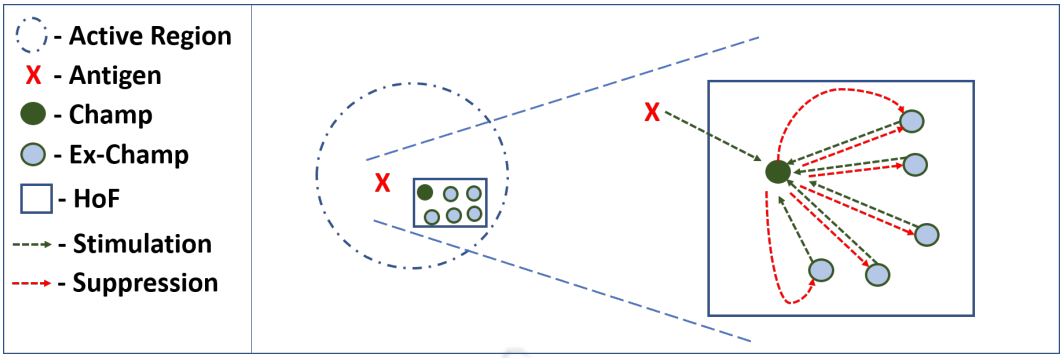


Figure 2.3: Idiotypic Network formed by antibodies within an *HoF*

$$\Delta C_{\nabla_i} = S_{Ag} + \xi * \sum_{j \in \nabla, j \neq i} S_{\nabla_j} \tag{2.4}$$

Here i is the index of the best antibody in the set ∇.

The concentrations of all other antibodies other than the best antibody in the ∇ which receive *suppressions* from the best antibody change as per the equation below:

$$\Delta C_{\nabla_j} = S_{Ag} - \xi * S_{\nabla_i} \qquad \forall j \in \nabla \; and \; j \neq i \tag{2.5}$$

This results in a decrease in the concentrations of the candidate antibodies as compared to the best antibody, due to the suppressions received from the latter.

The final concentration of an antibody after stimulations and/or suppressions is given by:

$$\forall k \in \nabla, \quad C'_{\nabla_k} = C_k + \Delta C_{\nabla_k} \tag{2.6}$$

Here, ΔC_{∇_k} denotes the change, either increase or decrease as per stimulation or suppression, in the concentration of the k^{th} antibody in the set of candidate antibodies, ∇.

2.4.3 Hall-of-Fame of Antibodies

The best performing antibodies generated in order to tackle antigens within an active region χ, are preserved in the HoF_{χ}. Thus, there are as many *Halls of Fame* as there are active regions, each containing the best set of antibodies specific to the associated antigens in that active region. All the antibodies present in the HoF_{χ} of a particular χ, form the set of *candidate antibodies* viz. ∇. These antibodies can thus deal with the antigens that crop up within the associated active region, χ. If the current best antibody, having dealt with the antigen fails, i.e., if its fitness drops below other antibodies in the repository, then one of the antibodies from the *HoF* is selected as the next best antibody to deal with the antigen. The candidate antibody which has been consistent in the past needs to be selected. The value of the concentration of an antibody indicates the level of consistency in performance of the antibody, in the past. A higher value of concentration would mean the antibody received more stimulations and less suppressions from the antigen and other antibodies over the past period. One could thus assume that such an antibody with high concentration has been performing consistently well. Antibodies with the high fitness values but low concentrations are the ones who have qualified into the *HoF* merely by performing well against an antigen a few times and hence cannot be deemed to be consistent. Thus, merely having a higher fitness value does not necessarily mean that the antibody is the best. It is for this reason that an antibody is selected which has highest concentration from those in the *HoF* to replace the current best whenever the latter exhibits a drop in fitness. The current best antibody is designated as the *Champ*, the antibodies other than the current best which are resident in the *HoF* as *Ex-Champs* and the offsprings which are generated as *Mutants*. The *Champ* and all the *Ex-Champs* constitute an *HoF*.

An antigen stimulates not only the *Champ* but also all the *Ex-Champs* in the associated *HoF* as per the Eqn. 2.1. The *Champ* and the *Ex-Champs* stimulate or suppress each other as per the Eqn. 2.3 - 2.6. The concentration of the *Champ* and all the *Ex-Champs* in the associated *HoF* changes due to stimulations and suppressions, as per the Eqn. 2.4 and 2.5 respectively. These *Ex-Champs* and the *Champ* form an *Idiotypic Network* where the *Champ* is stimulated by all the *Ex-*

Champs and all the *Ex-Champs* are suppressed by the *Champ* of the associated *HoF*. This network of antibodies within an *HoF* that stimulate and suppress each other, thereby forming an Idiotypic Network within the associated *HoF*, is portrayed in the Figure 2.3. Since, this Idiotypic Network is local to the respective *Halls of Fame*, they are termed as *Local Idiotypic Networks*. The overall search space thus comprises several active regions each of which houses an *HoF* within which an Idiotypic Network is conceived and maintained. Therefore there are as many Idiotypic Networks as there are *Halls of Fame* and active regions.

2.4.4 Eviction of Ex-Champs from *HoF*

It is highly likely that the antibodies that perform well against antigens are kept on adding to the *HoF*, the same could soon be over-populated. An overpopulated *HoF* would mean an increase in comparisons to be made in order to choose the best antibody (or *Champ*), which subsequently would result in a higher number of computations. A mechanism to contain and regulate the number of antibodies in the *HoF* is thus mandatory. A naive method would be to fix an upper limit to this number *a priori* constituting a hyper parameter. Since the antigens in each active region are bound to have different epitopes, the number of *Ex-Champ*s generated within each active region is bound to be different. This implies that there would be as many such hyper parameters as there are *Halls of Fame*. Fixing their values *a priori* would naturally be a tedious affair.

The concentration of an antibody could be used as a parameter to decide on its eviction from its *HoF*. However, there could be a case when an antibody Ab_i, coming into the *HoF*, has high fitness but lower concentration (due to low stimulations) than those already resident in the *HoF*. This high fitness and low concentration value would mean that though Ab_i exhibited better fitness it could be evicted earlier than the others in case of overpopulation of the *HoF*. Another approach could be to use fitness as the parameter to decide eviction. Fitness relates to the performance of an antibody with a specific antigen within an active region. In short, the fitness of an antibody varies with the antigen. Thus, an antibody could have a high fitness when it is selected to join the *HoF*. It may have low fitness for other antigens within the same χ. Under such conditions, better antibodies may be evicted earlier. Both

fitness and concentration, thus turn out to have their own disadvantages in becoming a measure to decide eviction of antibodies from an *HoF*.

Resource based Eviction

In this work, a quantum termed *Resource* (R) is defined, similar to the one described in [50], conferred to each antibody when it first enters its associated *HoF*. *Resource* could be visualized as virtual energy provided to an antibody. In living beings, energy is consumed whenever work is done while it is replenished when food is consumed. *Resource* is a similar parameter that is always consumed when the antibody tackles an antigen. It is replenished based on its current *Resource* and the current and past fitness values of the antibody. A quantum of *Resource*, R is initially conferred onto an antibody the very first time it is made to enter its associated *HoF*.

The *Resource* value at time *(t+1)* denoted by R_{t+1} is calculated as:

$$R_{t+1} = R_t + \Delta R \tag{2.7}$$

where ΔR is the change in the *Resource*, given by,

$$\Delta R = R_{rep} - R_{con} \tag{2.8}$$

$$R_{con} = \lambda_{mut} * (R_t / R_{Max}) \tag{2.9}$$

$$R_{rep} = \lambda_{ch} \tag{2.10}$$

where R_{rep} is the *Resource* replenished, R_{con} is the *Resource* consumed, λ_{mut} is the moving average difference in the fitness values of the *Champ* and the evolving *Mutants* and the λ_{ch} is the moving average difference in the fitness values of the *Champ*, over a defined window of generations. R_{Max} is the maximum value of the *Resource* which is initially conferred to the antibody upon inception in the *HoF*. The term R_t/R_{Max} ensures that the amount of consumed resource does not exceed the existing current resource.

An antibody is evicted from the associated *HoF* as and when its *Resource* becomes zero.

Ageing of Ex-Champs

It may be noted that the *Resource* of an antibody is consumed and replenished only when it is selected as the *Champ*. The *Ex-Champs* in the *HoF* have high *Resources* since their fitness values were high when they were selected as the *Champ* in the past. Since their *Resource* values remain constant and positive, the *Ex-Champs* will never be evicted, resulting in overpopulation within the *HoF*. In order to circumvent this problem, an *ageing factor* is introduced for each of the *Ex-Champs* in the *Halls of Fame*. This factor effects a decay of the *Resource* of the *Ex-Champs*, similar to that described in [50], so that those which are not selected as *Champs* for a considerable amount of time will eventually die out when their *Resource* values become equal to zero. Ageing of *Resource* is governed by the equation given below:

$$R_{t+1} = R_t - \exp^{\eta} \tag{2.11}$$

where R_{t+1} and R_t are the values of the *Resource* of an antibody at time $(t+1)$ and t respectively and η is a constant $(0 < \eta \leq 1)$. The *Resource* value of every *Ex-Champ* in the *HoF* is updated as per Eqn. 2.11.

2.4.5 iAES-HoF Algorithm

The proposed Immuno Inspired Action Evolution cum Selection algorithm with Hall-of-Fame *(iAES-HoF)* is portrayed in Algorithm 1. The vector of values sensed by the sensors on-board the robot forms the epitope (Ep) of the antigen (Ag). The best matched HoF_χ, the cross-reactivity threshold, ϵ, taken into consideration, is determined by comparing the affinity of the epitope with those of the *Champs* within each of the *Halls of Fame*. The number of comparisons to find the best matched HoF_χ is equal to the number of *Halls of Fame* in the system. Further, to find the *Champ* resident in the selected HoF_χ, a few additional number of comparisons may be required within.

The *Champ* for the next generation is then chosen based on an evolutionary strategy discussed later. While the antigen is made to stimulate the *Champ* and all the *Ex-Champs* present in this best HoF_χ, the *Champ* is made to suppress all the

Algorithm 1: *iAES-HoF* Algorithm

```
1  ϵ ← Constant;
2  MaxGenerations ← Constant;
3  MaxEvaluations ← Constant;
4  while MaxGenerations > 0 do
5      MaxGenerations ← MaxGenerations − 1;
6      Ep ← senseEnv();
7      Determine the best matched HoF_χ using Ep
8      if no HoF matches then
9          Ab_New ← createNewAntibody(Ep);
10         HoF_χ ← Ab_New;
11     Champ_HoF_χ ← getBestAb(HoF_χ);
12     call_Ag_Stimulation(HoF_χ);
13     call_Ab_Sti_Supp(Champ_HoF_χ, HoF_χ);
14     while MaxEvaluations > 0 AND Ag space is within χ do
15         MaxEvaluations ← MaxEvaluations − 1;
16         Evolve the Champ_HoF_χ by any Evolutionary Algorithm
17     consumeResource(Best_Ab_HoF_χ);
18     replenishResource(Best_Ab_HoF_χ);
19     for each Ab_i in HoF_χ do
20         if Ab_i ≠ Best_Ab_HoF_χ then
21             reduceResource(Ab_i)
```

Ex-Champs within HoF_χ, which thereby results in a change in the concentrations of the *Champ* as also all the *Ex-Champs*. The *Resource* of the *Champ* is consumed and replenished as per Eqn. 2.9 and 2.10 respectively. In addition, the *Resource* values of all *Ex-Champs* residing in HoF_χ are reduced as per the Eqn. 2.11. If the affinity between the epitope (Ep) of the current antigen Ag and the paratopes (Pt) of the *Champs* of the various *Halls of Fame*, is beyond the cross-reactivity threshold, ϵ, then a new antibody is created randomly and its paratope, Pt, is aligned to the epitope, Ep, of the antigen, Ag. A pre-determined quanta of resource and concentration are conferred to this newly created antibody before adding it to a newly created *HoF*. A new *HoF* essentially means the creation of a new active region χ.

To select the *Champ* from the *HoF*, tournament selection is employed. The selection is based on the values of the concentration. Out of the selected antibodies, the one with the highest concentration value is selected as the *Champ* to deal with the current antigen. If the fitness score is used as the criteria to select the *Champ*

then those antibodies who had a momentarily higher fitness would get selected as the *Champ*. Since its performance would not have tested over generations, it may or may not do well against the antigen. Hence, selection of the *Champ* based on only fitness score is not a correct strategy. The aim is to select the *Champ* which has performed fairly well against the antigens for a fair amount of time. Hence, considering the concentration value as the criterion for selection of the *Champ* fits this requirement.

If the antibody which has the highest concentration value in the *HoF* is selected, then the ones which have just entered the *HoF* which have lower concentration values currently would not get the chance to be the *Champs*. Since they may not get more chances, they may be evicted from the system. Therefore, tournament selection is employed using the concentration value to select the *Champ*, thus giving a chance to the antibodies which have low concentration values as well.

The computational complexity of the proposed algorithm is analyzed and found to be $O(n)$. In line number 7, determining the best matched HoF_χ has the complexity of $O(n)$. Similarly, line numbers 12, 13, 17 and 18 which are all functions that are called by the algorithm, exhibit the same order of complexity. The loop in line number 14 and the process in line 16 have the computational complexities $O(1)$ and $O(n)$, respectively. The loop in line number 19 also exhibits a computational complexity of $O(n)$ since there is a finite number of antibodies in an HoF_χ. Taking into consideration all these, the overall computational complexity of the Algorithm 1, comes out to be $O(n)$. The main loop of the algorithm (line number 4) is executed for a constant *MaxGenerations* number of times, and hence does not play a role in the above complexity.

In this work, (1+1) Online EA [14] is used to evolve the best controller viz. the *Champ*. The *Champ* is mutated, with a random probability, using the Gaussian mutation function $N(0,)$, to generate a *Mutant*. The fitness of the *Mutant* is determined based on its performance. Otherwise (if the *Champ* is not mutated), the same is done for the *Champ*, which either becomes an *Ex-Champ* or continues to be the *Champ* based on its fitness. If the fitness of the *Mutant* is greater than the *Champ*, then the *Mutant* becomes the *Champ*, while the latter becomes an *Ex-Champ*. Both of them are added on to the associated *HoF*. If not, the *Mutant* is discarded and the

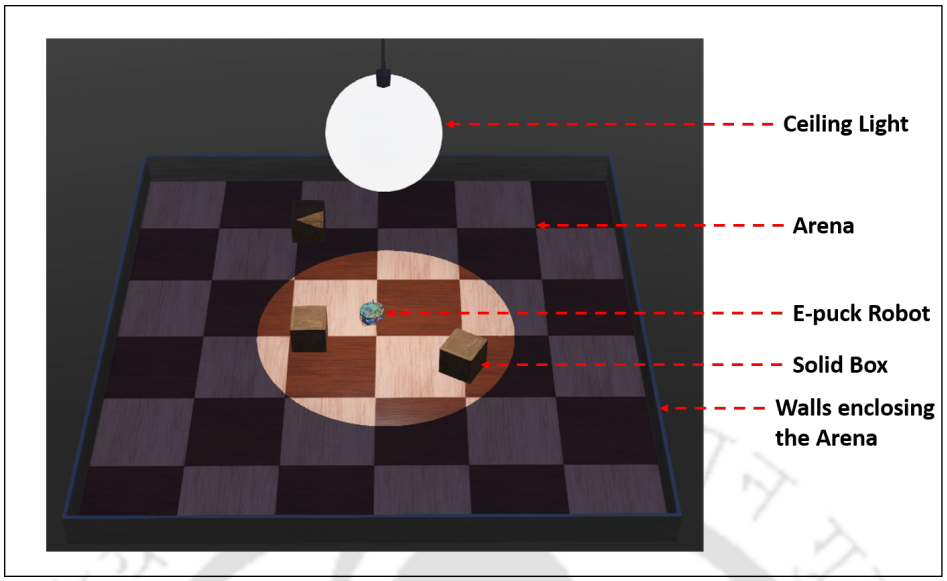


Figure 2.4: Arena in the *Webots* simulator

value of σ is doubled for use in the next generation. This doubling of σ is continued till its value reaches a σ_{max} value, after which it is set to a minimum and the process is continued. In this method, only the mutation strategy is used and not crossover.

The tuning of the ANN-based controllers is done by Gaussian mutation. The mutational process introduces randomness to the evolution, thereby equipping the proposed method to handle uncertainty well. This also prevents the evolving controller from getting stuck in the local optima. Also, since the active regions are being evolved dynamically without any external intervention, the proposed algorithm handles any uncertainty in the search space robustly. It is also quite robust to handle any change or addition in the search space. It creates a new active region if such a space is not encountered before. When it comes to maintenance of this active region, the *Concentration* and *Resource* parameters take care of the re-selection and eviction of the controllers from the *HoF* of this active region. Also, controllers in each *HoF* are evolved to cater to varied scenarios in the associated active region.

2.5 Experimental setup

The experiments are performed using the two algorithms *iAES* and *iAES-HoF* both on a simulator and a real robot. Simulations were carried out mainly to explore and comprehend the nature and characteristics of the algorithms and enable multiple

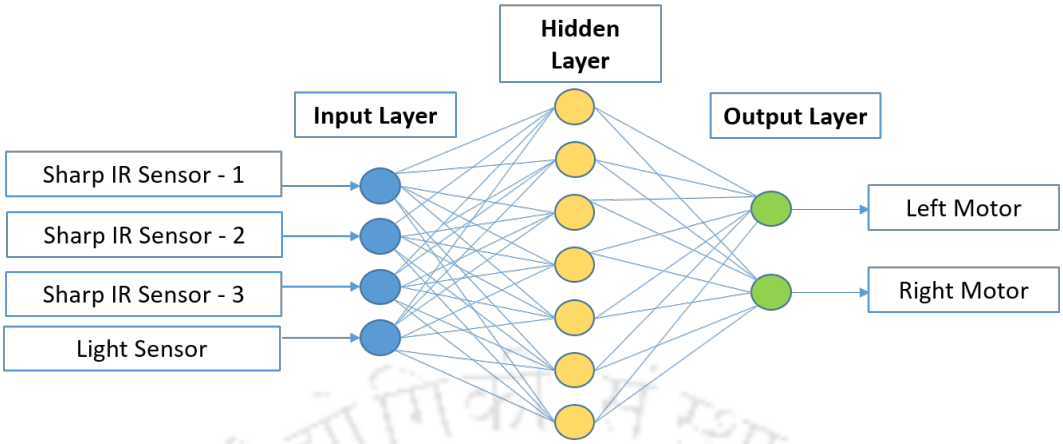


Figure 2.5: An Artificial Neural Network as a Robot Controller

runs. Since the real world always forms the best and ultimate test bed for evaluating the efficacy of an algorithm, experiments were also performed using a real robot. This provided a means to assess the performance of the algorithm in the real world. Prior to presenting the results of the experiments performed, the simulation and real world experimental setups are described.

2.5.1 Simulation Setup

A simulated e-puck robot available within the *Webots* [87] simulator was used to test the algorithm under simulation. Figure 2.4 depicts a simulated e-puck in an arena. Three out of the eight IR sensors on-board the e-puck are used to measure the distance of the robot from an obstacle. Higher the values reported by the IR distance sensor, closer is the robot to an obstacle while lower values indicate an obstacle-free area. The e-puck also has IR based light sensors to measure the intensity of light. An IR light sensor is used to measure this intensity during Phototaxis. Higher light sensor values indicate higher intensities of ambient light while lower values mean lower intensities of the same. The dimension of the arena was set to 1.5x1.5m with walls of height 0.1m height surrounding it. The walls and the three squared solid boxes within the arena seen in Figure 2.4, acted as obstacles for the robot. A simulated ceiling light suspended at the center of the arena within Webots, facilitated as the light source to be used for Phototaxis.

As explained in section 2.4.1, the vector formed by the values reported by the four sensors used, constituted the *Ep* of the *Ag*. This work uses an Artificial

Neural Network (ANN), as shown in the Figure 2.5, as the controller *Ctr* within an antibody which actually drives the robot. Since there are four sensory inputs, the ANN too has four input nodes. As can be seen, the ANN has seven hidden nodes and two output nodes interfaced to left and right motors of the robot. The Hyperbolic tangent (tanh) function was used for activation.

The proposed *iAES-HoF* algorithm, *iAES* algorithm and (1+1) Online EA were tested for the tasks of Obstacle-avoidance (OA) for 400 generations and Phototaxis-cum-obstacle-avoidance (PT-OA) for 600 generations using the simulated e-puck robot. In the former task, the robot when placed at any random location in the arena, was supposed to learn to move around the arena while also avoiding the obstacles. In case of PT-OA task, the robot was required to learn to move towards the source of light while also avoiding obstacles encountered in the process making it a more complex task than OA. It may be noted that in the first task (i.e. mere Obstacle-avoidance) the task of avoiding the obstacles was to be learned in the absence of light while in PT-OA, the inherent Obstacle avoidance needed to be learnt in the presence of light.

After initial experiments for different values of ϵ in the simulated environment, it was found that the robot could learn to perform PT-OA for values ranging from 0.3 to 0.65. Thus, the ϵ values 0.3, 0.45, 0.6 and 0.65 are used for the comparative study. The following equation was used to determine the fitness, F, of the controller (either *Champ* or *Mutant*), associated with the task:

$$F = \sum_{t=0}^{T} (f_O + f_P) \tag{2.12}$$

$$\text{where } f_O = v_{tr} * (1 - v_{ro}) * d,$$
$$d = \min_{1 \leq i \leq 3}(1 - IR_i)$$

where f_O and f_P are the fitness functions associated to the tasks of OA and PT-OA, respectively. f_O was adapted from [96]. v_{trans} and v_{rot} are the translational and rotational speeds of the robot, respectively, while d is the minimum distance of the robot from the obstacle. d was taken to be the minimum of three values reported by the three IR distance sensors on board the e-puck robot. f_P is the normalized

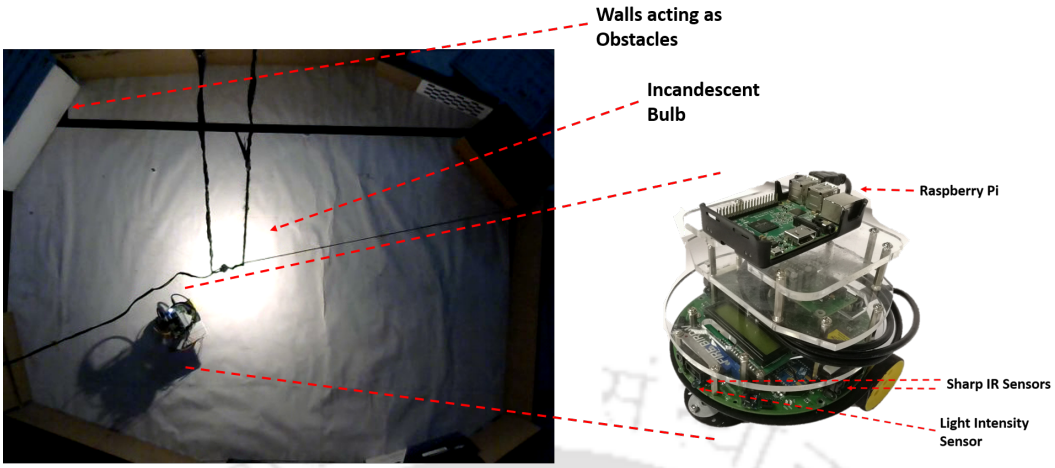


Figure 2.6: The Arena and the Firebird Robot used for experiments

value of the intensity of light obtained from the light sensor. The intensity obtained varied from 0 (dark) to 1 (bright). Here, a higher value of fitness value F is tried to obtain. When there is no obstacle in the path of the robot, the value of d is high. Under such a condition, the robot needs to increase v_{tr} and decrease v_{ro} so as to obtain high fitness f_O. When an obstacle is encountered, d is low which makes the robot take the opposite decisions. Similarly,in the presence of light, the value obtained by light sensor is high, thereby resulting in an increase in the value of f_P.

The parameters used in the simulation experiments were as follows: $\alpha = 0.3$, $\beta = 3$, Initial concentration of $Champ = 10$, $R_{max} = 20$, Window size of $\lambda_{ch} = 5$, Window size of $\lambda_{mut} = 10$ and $\eta = -0.6$. The value of σ_{max} was set as 4. The number of evaluations of the antibody (as in the algorithm 1) was set as 20, corresponding to 6 seconds. α and β are constants merely used for magnifying the values within the equations 2.1 and 2.3.

The robot in the arena samples the environment through its sensors, and the values constitute the *Ep* of *Ag*. The corresponding *Ab*, whose paratope matches with this *Ep* is used to tackle *Ag*. As mentioned earlier, the *Ab* comprises, among other items, a set of weights which are used by the ANN based controller (*Ctr*).

2.5.2 Real Robot Setup

In order to understand how well the algorithm works in the real world, a Firebird V [1] robot is used for the real experimentation. This robot was equipped on-board, with three IR sensors and a light sensor, as shown in Figure 2.6,. The $Sharp^{TM}$ IR sensor was used to measure the distance of an obstacle from the robot. The range of the sensor varied from 0-800mm. The light sensor values varied from 0-255, with the lower values indicating that the robot is in the vicinity of a brighter light source. Just as in the simulated robot experiments, the vector formed by the values reported by these four sensors, constituted the Ep of the Ag. The same Artificial Neural Network (ANN) having the same topology shown in the Figure 2.5 was used as the controller Ctr within an antibody. A 2mx2m sized arena (shown in Figure 2.6) with an incandescent bulb suspended at the center to facilitate the task of Phototaxis, was used. The arena was enclosed by walls, which acted as the obstacles.

Just as in the simulations, the proposed algorithms were tested in the real-world by making a real robot learn both the tasks of OA and PT-OA. However, the results of the former task are not portrayed, as the same were very much in line with those obtained in simulated environments and hence did not reveal anything new. Just as in the simulations, the proposed algorithm was tested for the task of Phototaxis-cum-obstacle-avoidance on part of the robot for 300 generations within the above-mentioned arena. When placed in any random location in the arena, the robot needed to learn to move towards the source of light while also avoiding obstacles encountered in the process.

The same ϵ values 0.3, 0.45, and 0.6 are used, as in the simulations, for comparison. The fitness function, F, associated with the task was calculated using the Eqn. 2.12. Since the nature of the values reported by the sensors on-board the Firebird V robot is different from that of the simulated robot, the terms $dist$ and f_P in Eqn. 2.12 were calculated as given below:

[1] More information about the Firebird V robot can be found at http://nex-robotics.com/products/fire-bird-v-robots/fire-bird-v-atmega2560-robotic-research-platform.html

$$d = \min_{1 \leq i \leq 3} (IR_i)$$

$$f_P = 1 - L$$

Here, d is the minimum distance of the robot from the obstacle. L is the normalized value of the Light sensor, where 0 corresponds to to the brightest light and 1 corresponds to no light. The value of f_P is higher when there is a presence of light and vice versa when the intensity of light is low. These two values were substituted along with other determined values in the Eqn. 2.12 to find the fitness F. As in the case of the simulated setup, in the experiments using the real robot too, a high fitness value is aimed at.

As in simulation, here too, the real robot placed in the arena, samples the environment through its sensors. These sensor values constitute the *Ep* of *Ag*. The corresponding *Ab*, whose paratope matches with this *Ep* is used to tackle *Ag*. As mentioned earlier, the *Ab* comprises, among other items, a set of weights which are used by the ANN based controller i.e. *Ctr*. All the related parameters in the real robot experimental setup were set to those mentioned in the simulation setup. In doing so, it is ensured that the simulated and the real setup based experiments were performed under almost the same conditions.

2.6 Results and Discussions

In this section, the results obtained by running the *iAES* and the *iAES-HoF* algorithms in both simulated and real worlds are discussed. The effect of caching *Champs* in an *HoF* is also presented followed by the effects of varying the value of ϵ in both these worlds. Graphs that bring out the significance of the concepts of *Concentration* and *Resource*, in the context of selection of *Champs* and eviction of *Ex-Champs* from the *HoF*s have also been presented. A total of 10 different runs were performed in the simulation and 3 runs in the real world setup. In each of these runs, the starting position of the robot was set to different positions. In all

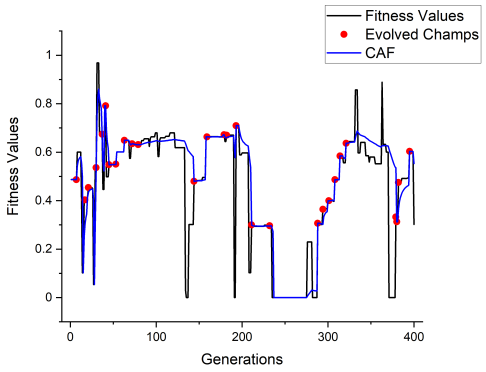


Figure 2.7: Evolution of Champs in (1+1) Online EA for the OA task in the simulated world

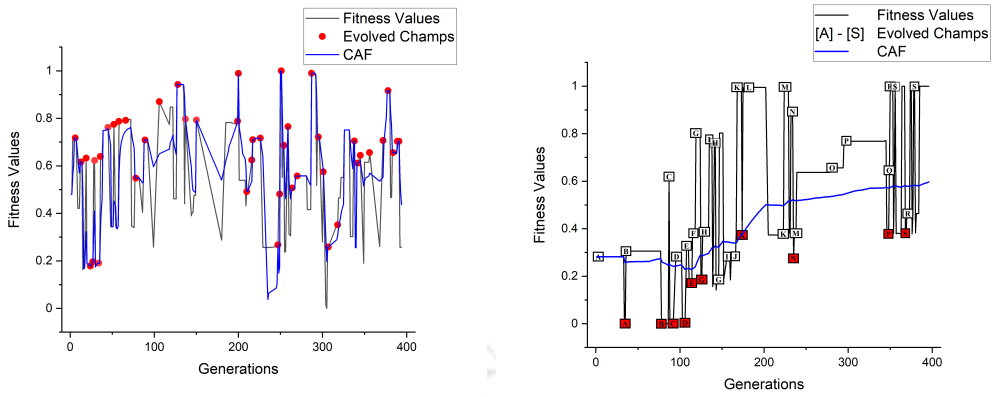
of these runs similar results were observed, hence one of these runs are protrayed in the results. To portray the results, the fitness values and the Cumulative Average Fitness (CAF) are plotted versus the generations. In the case of *iAES*, where there is only one *Champ* in the *HoF*, the CAF is taken as the average of the fitness values of the *Champ* over the number of generations in which it has retained its position as the *Champ* before eventually being evicted. In case of *iAES-HoF*, the CAF is the average of the fitness values of the current *Champ* from the instant of time it was inducted into the *HoF*. Thus, in both the cases of *iAES* and *iAES-HoF*, the CAF value reflects the fitness of the *Champs* over generations till its final eviction.

2.6.1 Effect of Caching *Champs*

The effect of caching *Champs* is described from the studies conducted in the simulated world using both *iAES* and the proposed *iAES-HoF* algorithms with various ϵ values. In addition, both the algorithms are compared with the (1+1) Online EA.

Caching *Champs* - Simulation Results

As discussed earlier, since the search space in the (1+1) Online EA is not divided, it evolved only one generic controller for each of the tasks - OA and PT-OA. On the contrary, in case of the *iAES* and *iAES-HoF* experiments, since several active regions were formed, many such controllers were evolved for each of these tasks. Therefore, only that dominant active region is considered which encountered the



(a) (b)

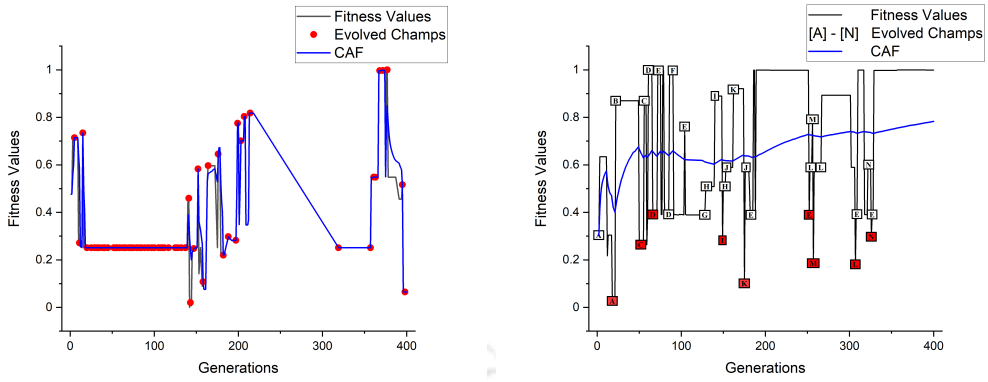
Figure 2.8: Evolution of Champs in an Active Region with $\epsilon = 0.3$ for the OA task in the simulated world with (a) *iAES* and (b) *iAES-HoF*

maximum number of attacks from the respective antigens. To ensure that the comparisons are fair, both the arena and the robot's starting positions, chosen randomly, were kept the same for all the sets of experiments. Hence, the resulting dominant active region also remained the same. The figures 2.7 - 2.15 portray the changes in fitness values over generations.

In the graphs of the *iAES-HoF*, each letter indicates a *Champ*. If the same letter appears repeatedly in the graph, it implies that the previous *Champ's* fitness was lower due to which this particular *Champ* was activated from the respective Hall of Fame. A red coloured letter indicates that this evicted *Champ* was put back into the Hall of Fame. However, when the eviction and reappearance of the *Champs* are closely spaced in the generations, it was difficult to explicitly indicate the same in the graph. Hence, the appearance of a different *Champ* in the graph implies that the previous one was evicted.

Numerous Champs evolved in cases of both the (1+1) Online EA and the *iAES* algorithm. Since these could not be indicated using the English letters, the evolution of the newly evolved Champs are portrayed using tiny red circles in the graphs associated to these algorithms. These newly evolved *Champs* evicted their previous ones from the system.

The graph in the Figure 2.7 represents the change in the fitness values of the *Champs* versus the generations for the (1+1) Online EA. From the graph it can be observed that there are drastic rises and falls in the fitness values. As stated



(a) (b)

Figure 2.9: Evolution of Champs in an Active Region with $\epsilon = 0.45$ for the OA task in the simulated world with (a) *iAES* and (b) *iAES-HoF*

earlier, since the search space was not divided into different regions, this strategy tried to evolve only a single generic *Champ* for whole of the search space, which causes drastic fall in the fitness values when the robot moved to a space where the current *Champ* had not learned to perform well. It was observed in the experiments that the robot could not learn the OA task effectively in whole of the space.

The graphs in the Figure 2.8a and Figure 2.8b represent the change in the fitness values of the *Champs* versus the generations of evolution of antibodies in case of *iAES* and *iAES-HoF* algorithms respectively, for an ϵ value of 0.3. It can be seen that several antibodies evolve when the *iAES* algorithm was used, as indicated by the tiny red circles in Figure 2.8a. Further, the fitness values rise and fall over the generations since the better performing mutants tend to replace the current *Champ*. Unfortunately, this replacement also takes place for even a consistently well-performing *Champ* when its fitness falls transiently below that of its *Mutant*. The erratic changes in fitness are mainly because the new *Champ* does not always perform well in the whole of the active region. Its fitness value thus often drops in the next generation, giving rise to another inexperienced *Mutant* which displaces the current *Champ*. On the contrary, in case of the *iAES-HoF* algorithm, the *Champ* re-enters the hall of fame, whenever it is replaced by a *Mutant*. When the *Mutant's* performance falls, an *Ex-Champ* from the Hall of Fame takes over as the *Champ* again. This greatly avoids drastic falls in the fitness values as observed in the graph in Figure 2.8b). Thus, a comparatively stable CAF is attained in the case

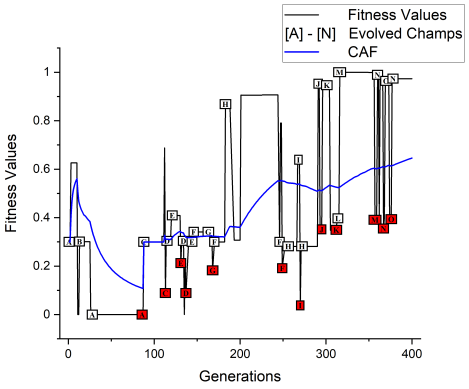


Figure 2.10: Evolution of Champs in an Active Region with $\epsilon = 0.6$ for the OA task in the simulated world with *iAES-HoF*

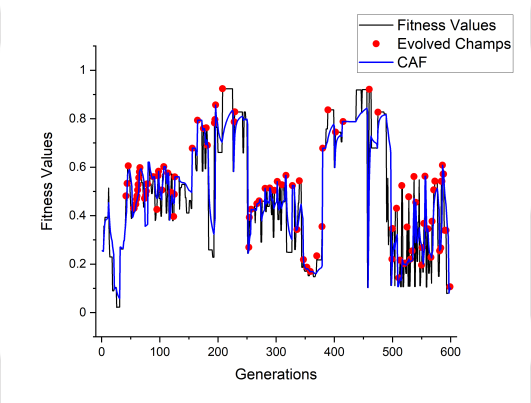
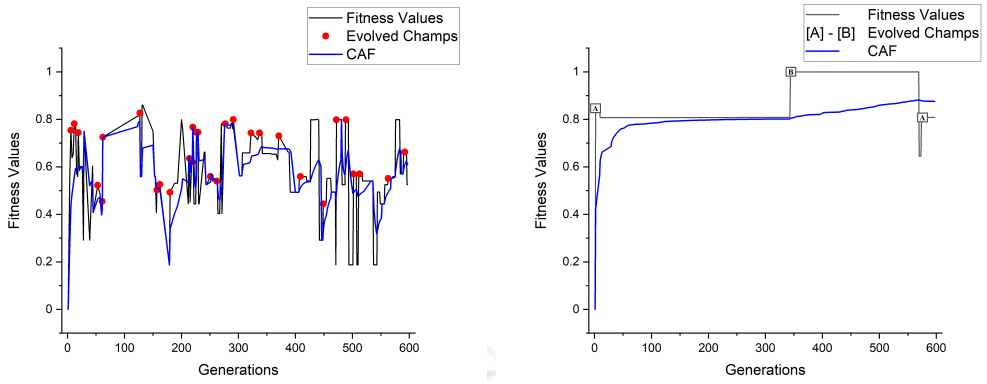


Figure 2.11: Evolution of Champs in (1+1) Online EA for the PT-OA task in the simulated world

of the *iAES-HoF* algorithm, as opposed to chaotic variations seen when the *iAES* algorithm is used.

The graphs in the figures 2.9a and 2.9b represent the trends in fitness values for the task of OA in case of for *iAES* and *iAES-HoF*, respectively with an ϵ value of 0.45. Here, the fitness values are on the lower side for *iAES* as compared to that when the *iAES-HoF* is used. The higher ϵ value of 0.45 results in creation of larger active regions. This means the evolving controllers need to perform well in the whole of such active regions. Unfortunately, the *iAES* was not able to evolve one single controller per active region which could cater to whole of such regions, which led to the drastic rises and falls in the fitness values as seen in the Figure 2.9a. On the contrary, in the *iAES-HoF*, a stable learning curve with a higher CAF can be observed in case of *iAES-HoF*, due to the caching of the *Champs* and their



(a) (b)

Figure 2.12: Evolution of Champs in an Active Region with $\epsilon = 0.3$ for the PT-OA task in the simulated world with (a) *iAES* and (b) *iAES-HoF*

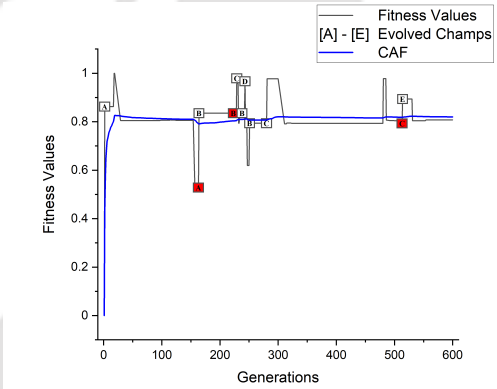


Figure 2.13: Evolution of Champs in an Active Region with $\epsilon = 0.45$ for the PT-OA task in the simulated world with *iAES-HoF*

subsequent re-use, in spite of creation of large active regions.

It was found that when simulations were carried out using *iAES* with the ϵ values greater than 0.45 the robot did not learn anything significant. Hence, no graphs pertaining to these have been presented. In the Figure 2.10, for the *iAES-HoF* with an ϵ of 0.6, the higher valued CAF curve suggests a fairly stable learning behaviour as compared to the case when the ϵ value was 0.45.

It may be noted that in contrast to the (1+1) Online EA and the *iAES* algorithm, the CAF curves of the *iAES-HoF* algorithm, for different values of ϵ stabilize indicating an enhancement in both learning and performance of the respective controllers.

The graph in the Figure 2.11 represent the variations in the fitness values versus

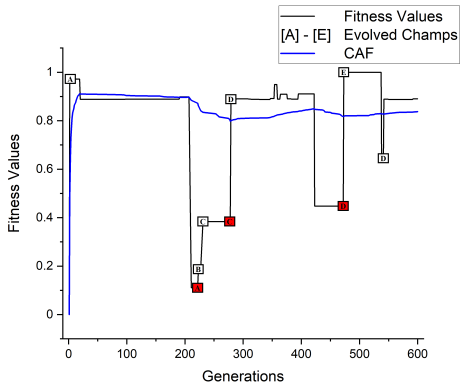


Figure 2.14: Evolution of Champs in an Active Region with $\epsilon = 0.6$ for the PT-OA task in the simulated world with *iAES-HoF*

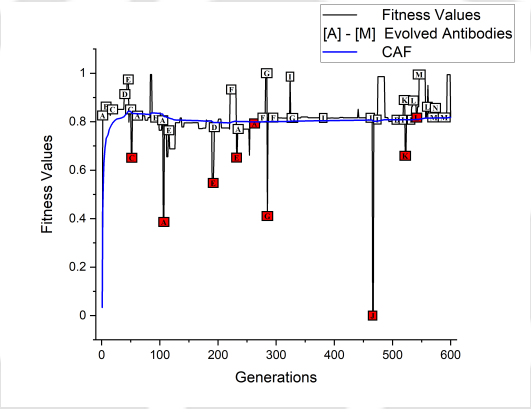
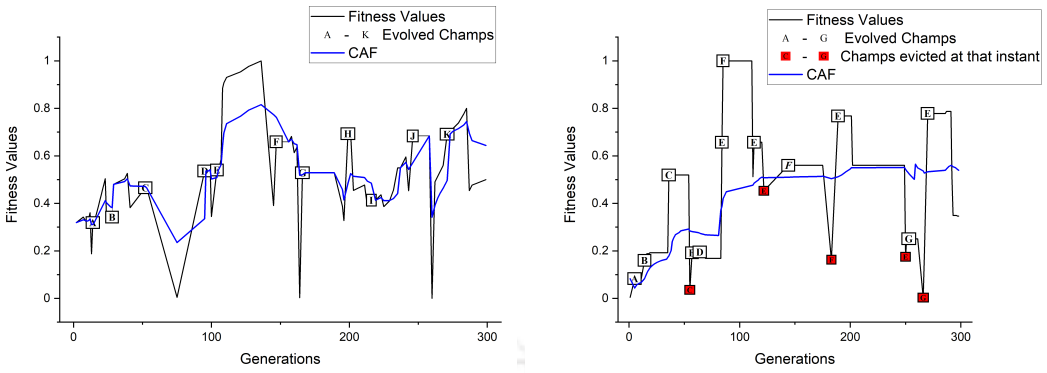


Figure 2.15: Evolution of Champs in an Active Region with $\epsilon = 0.65$ for the PT-OA task in the simulated world with *iAES-HoF*

the generations for the task of PT-OA for (1+1) Online EA. The *Champ* had higher fitness values in a certain subspace of the search space, but when the robot moved to other subspaces, there was a drop in the fitness values of the *Champ*. Even for this task, the robot could not learn the task due to the lack of evolution of a generic controller.

The graphs in the figures 2.12 - 2.15, depict the variations in fitness values versus the generations for the task of PT-OA when the algorithms *iAES* and *iAES-HoF* were used. As mentioned earlier, though PT-OA task is comparatively more complex than OA the graphs in Figure 2.12a and Figure 2.12b show similar trends as in the OA task when ϵ equal to 0.3. There are rises and falls in the fitness values in case of the *iAES*, while a stable learning pattern can be inferred when the *iAES-HoF* algorithm was used. Given the complex nature of the PT-OA task, the *iAES*



(a) (b)

Figure 2.16: Evolution of Champs in an Active Region with $\epsilon = 0.3$ for the PT-OA task in the real world with (a) *iAES* and (b) *iAES-HoF*

was not able to learn the task for ϵ values greater than 0.3. The *iAES-HoF*, on the contrary, was able to learn the task for the ϵ values 0.45, 0.6 and 0.65 with a stable learning curve as shown in the Figure 2.13, Figure 2.14 and Figure 2.15. Though an increase in rises and falls in the values of fitness is observed with the increasing ϵ values, it can be seen that the learning is stable as indicated by the CAF curve in case of the *iAES-HoF* algorithm.

For the values of ϵ greater than 0.3, the robot could not learn the task of PT-OA using the *iAES* algorithm, whereas in the case of *iAES-HoF*, it was able to do so. This is because the gradient of the light intensity in the darker areas is very low (as shown in Figure 2.4). It is thus difficult for the same controller (or *Champ*) to learn to adapt to both very low and high light intensity gradients meaning thereby that more such controllers/antibodies are required. The *iAES-HoF* algorithm circumvents this drawback by making use of a host of controllers/antibodies (*Champs* and *Ex-Champs*) within the *HoF* that can cater to such varied conditions.

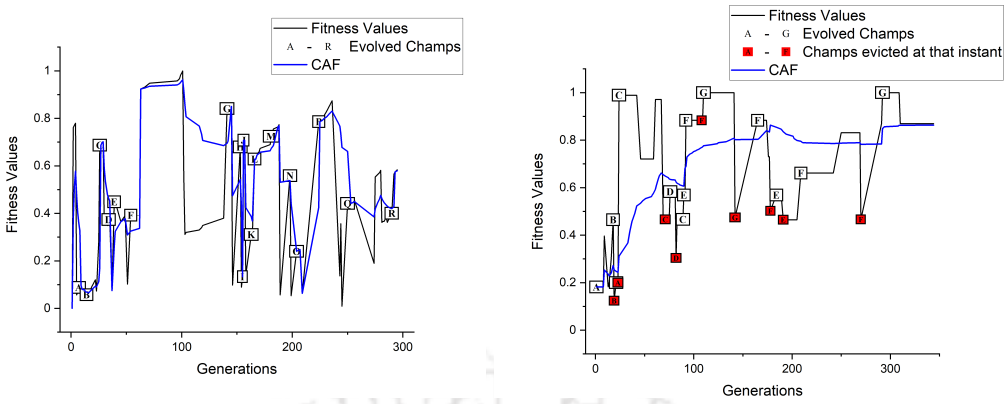
The experiments performed in the simulated world indicate that the use of the *iAES-HoF* algorithm results in a more stable learning behaviour with higher CAF values as compared to those obtained when (1+1) Online EA and *iAES* was used. The simulation results thus, validate and endorse the need to cache the *Ex-Champs*.

Champs - Real World Results

As stated earlier, both the tasks OA and PT-OA are performed using *iAES* and *iAES-HoF* algorithms running on a real robot. The controllers produced during the learning of the former task showed trends which were similar to those obtained in simulation. Hence, in this section, only those for the task of PT-OA are highlighted. These tasks are not performed for the (1+1) Online EA as Semwal et al. have already established that their proposed algorithm (i.e. *iAES*) outperforms the (1+1) Online EA in real world experiments [110]. The corresponding graphs of fitness values versus generations for this task, are shown in Figure 2.16 - 2.18. The evolved antibodies both in the *iAES* and *iAES-HoF* are indicated by the English letters. Similar to the experiments in the simulated world, multiple active regions were evolved. For the sake of analysis, only that active region is considered which encountered the maximum number of attacks by the associated antigens. It may be noted that for both the algorithms, the environment and arena used were identical. Thus, the dominant active regions also remained the same in both the cases. The graphs depict the performance of the antibodies in the dominant active region. In the graphs, the horizontal axis depicts the generations for which the experiment was conducted, however it does not mean that only this particular active region was active throughout. Other active regions are also coming up which are not indicated in this graph.

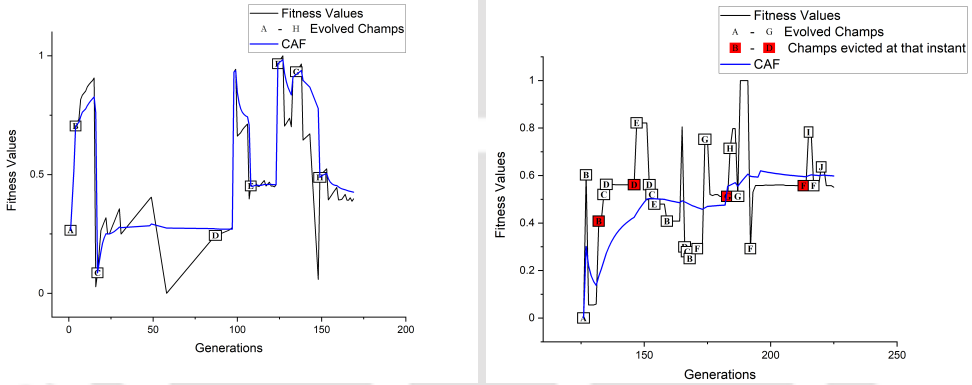
The Figure 2.16a and Figure 2.16b represent the change in fitness values over the generations of the antibodies in case of *iAES* and *iAES-HoF* algorithms, respectively, for an ϵ value of 0.3.

The various letters in the square brackets indicate the antibodies that evolved over the generations. It can be seen that in case of the *iAES* algorithm, as soon as a *Mutant* (for instance, [B] in Figure 2.16a) performs better than the current *Champ* ([A] in Figure 2.16a), the former replaced the latter. This happens even when the *Champs* have performed well for several generations. For example, the antibody [E] initially achieved a high fitness value but since it did not perform well for an antigen within its associated active region, its fitness dropped drastically. A *Mutant* [F] which performed better in the next generation, thus replaced the antibody [E]



(a) (b)

Figure 2.17: Evolution of Champs in an Active Region with $\epsilon = 0.45$ for the PT-OA task in the real world with (a) *iAES* and (b) *iAES-HoF*



(a) (b)

Figure 2.18: Evolution of Champs in an Active Region with $\epsilon = 0.6$ for the PT-OA task in the real world with (a) *iAES* and (b) *iAES-HoF*

as the *Champ*, causing the latter to be evicted though it had performed fairly well, earlier. It can also be seen that this new *Champ*, [F], which was well primed to tackle a specific antigen, was soon replaced by another newer *Mutant* [G] after a subsequent attack by a new antigen within the same active region. Similar events can seen to have occurred with other antibodies as well, resulting in instability in the learning exercise and hence the gradient. This is indicated by the rise and fall of the CAF of the *Champs* (blue line in Figure 2.16a) over the generations.

The experiments conducted with the *iAES-HoF* algorithm exhibited a comparatively stable and better learning paradigm. Since the past *Champs* are being cached, whenever a drastic fall in the fitness value of a *Champ* occurs (for instance,

[G] in Figure 2.16b)), an *Ex-Champ* in the associated *HoF* (for instance, [E] in Figure 2.16b) replaces the current *Champ*, unlike that in case of the *iAES* algorithm where a new and naive *Mutant* takes over as the *Champ*. In the proposed algorithm, instead of evicting the current *Champ* immediately (as in case of the *iAES* algorithm), it was cached in the associated *HoF*. Though rises and falls in the fitness values can be observed even in case of the *iAES-HoF* algorithm, the caching and re-use of the *Ex-Champs* with higher concentrations, resulted in a higher and more stable CAF (blue line in Figure 2.16b)).

The Figure 2.17a and Figure 2.17b represent the changes in fitness values of the antibodies over the generations for both the *iAES* and *iAES-HoF* versions of the algorithm, respectively, for ϵ equal to 0.45. For the *iAES* algorithm, the drastic rise and fall in the fitness values is more prominent than the previous case (when $\epsilon = 0.3$). More number of naive mutants also seem to have been evolved and replaced the corresponding *Champs*, without improving the fitness of the *Champs*. The CAF thus, also exhibits turbulence (blue line in the Figure 2.17a) causing degraded learning.

In case of *iAES-HoF* algorithm with an *epsilon* value of 0.45, there is no drastic fall in the fitness values. This is because, the *Ex-Champs* are re-selected as *Champs* from the *HoF* selected based on their concentrations. The learning seems to be more stable than that of the *iAES* algorithm, as indicated by the corresponding CAF curve (blue line in the Figure 2.16b).

Similar trends were observed when the ϵ value was set to 0.6. The Figure 2.18a and Figure 2.18b represent the changes in fitness values over the generations of the antibodies in case of *iAES* and *iAES-HoF* algorithms, respectively for an ϵ value of 0.6. In Figure 2.18b, initially for around 125 generations, the learning took place in one active region, after which it switched to the second more dominant active region. The graph in Figure 2.18b thus, commences from the 125th generation. It may be noted that the final value of the CAF was highest when ϵ was 0.45 while for the other two, viz. 0.3 and 0.6, the CAF had lower values.

As is clear from the discussions above, the results obtained from both simulation and real-world experiments seem markedly similar strongly indicating the viability of the proposed algorithm in practical scenarios. It may however be ob-

served that in the simulated world, the *iAES-HoF* could learn the PT-OA task in spite of the fact that the intensity gradient of light was not prominent in the darker areas of the arena (as seen in Figure 2.4). This was not so in case of the *iAES* algorithm. The studies above strongly indicate the efficacy of caching *Champs* in an HoF.

2.6.2 Effect of ϵ

ϵ divides the search space into several active regions. More the value of ϵ, lesser is the number of active regions and more is the hyper volume of each of these active regions. Higher hyper volumes indicate higher number of diverse antigens causing more number of antibodies to evolve within. Generating generic antibodies to cater to a high number of diverse antibodies becomes a difficult exercise. Higher values of ϵ can thus slow down the learning process.

Low hyper volumes resulting from lower values of ϵ have lower numbers of diverse antigens. Thus, only a few antibodies per active region, need to be evolved to cater to them. This may initially seem to increase the learning rate. However, every time an antibody is generated to tackle an antigen within a specific active region, the next antigenic attack that succeeds this event, invariably occurs at a neighbouring active region. Thus, the learning exercise is switched constantly from one active region to another which in turn slows down the learning rate. In terms of the two algorithms discussed herein, this could be explained based on the system constantly switching from one *HoF* of an active region to that of another. Such a switching tends to give lesser chances to an antibody to learn to tackle antigens within their respective active regions. Thus, dividing the search space using ϵ is critical while solving a problem.

The *iAES* algorithm endeavours to find one generic antibody per active region that can tackle all antigens within. Evolving such an antibody to cater to a diverse set of antigens is naturally difficult. Thus, increasing or decreasing ϵ does not ameliorate the learning problem. Lower values of ϵ will aid in evolving a generic antibody but the resulting increased switching between active regions (due to diverse antigenic attacks) tends to decrease the learning rate. Likewise, an increased value of ϵ, reduces the switching but also makes it difficult to evolve a generic antibody for

that active region. The *iAES-HoF* algorithm may also suffer from similar problems due to switching and diversity of antigens as in the *iAES* version. However, since *iAES-HoF* uses a set of generic antibodies, rather than merely one generic antibody, the chances of evolving this set to cater to most of the antigens associated to an active region is higher thereby improving the learning rate. In the simulated robot experiments, it was found that for ϵ equal to 0.3, 0.45 and 0.6, the numbers of active regions generated, were found to be: (a) For the task of OA - 7, 4 and 2 for the respective ϵ values (in both *iAES* and *iAES-HoF*). (b) For the task of PT-OA - 11 (in both *iAES* and *iAES-HoF*), 7 and 3 (in *iAES-HoF*) for the respective ϵ values. Since the tasks OA and PT-OA were of different complexities, their number of active regions also varied. In the real robot experiments, for the task of PT-OA, for ϵ equal to 0.3, 0.45 and 0.6, the number of active regions generated, in both iAES and iAES-HoF, were found to be 13, 5 and 2, respectively. Thus, in case when ϵ is 0.3 for the task of PT-OA, the relevant graphs in Figure 2.16a and Figure 2.16b show a drastic rise and fall in fitness values due to switching between several active regions, while when ϵ is equal to 0.6 (Figure 2.18a and Figure 2.18b), the system remained in the same active region due to its high hyper volume, in spite of a diverse antigenic attack. Here too, the rise and falls were prominent due to the diversity of antigenic attacks resulting in the production of newer antibodies with lower concentrations. With ϵ equal to 0.45 (Figure 2.17a and Figure 2.17b), the number of active regions was moderate thereby lowering the switching between such regions. Consequently, antibodies within an active region were exposed to the associated antigens more frequently before a switch to another active region. In case of the *iAES-HoF* algorithm, this aspect increased the chances of antibodies within an active region evolving into better ones resulting in an enhanced learning process. The higher values of CAF especially in the case when ϵ was set to 0.45 indicate this enhancement in learning.

Further, to determine if there is a statistically significant difference between the *iAES-HoF* and *iAES* algorithms, the non-parametric Mann-Whitney U test [85] with different ϵ values for both the simulated and real-world experimental runs were conducted. The significance level for all the runs were set at 0.01. In the simulations, for the task of OA, the p-values were found to be 0.00002 and 0.00001

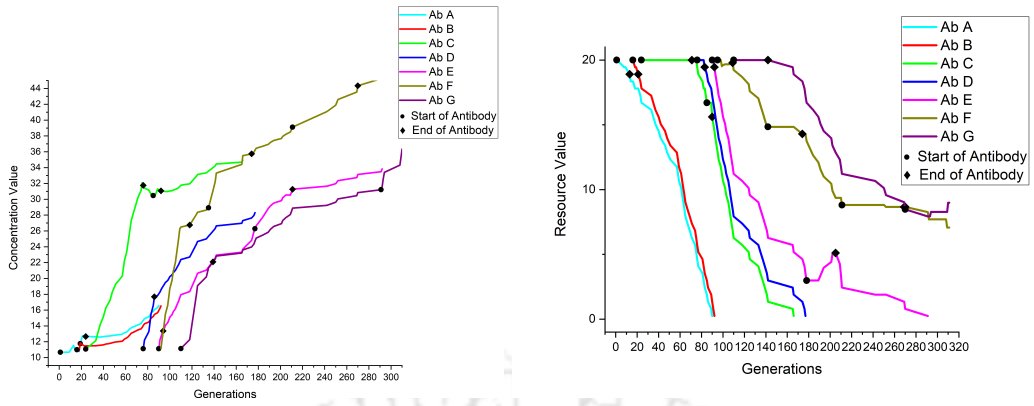
for the values of the epsilon 0.3 and 0.45, respectively. Similarly, in the real world experiments, the p-values, for the PT-OA task, obtained with the significance level of 0.01 were found to be 0.0002, 0.00001 and 0.0009 for the values of ϵ 0.3, 0.45 and 0.6 respectively. The Mann-Whitney U test was not conducted for those ϵ values for which the *iAES* failed to learn the tasks. All the p-values from the tests, in both the simulation and real-world experiments, were found to be less than the significance level set as 0.01. This shows that for all the runs the results are statistically significant and not just stochastically variant.

2.6.3 Significance of Resource

The value of the concentration of an *Ab* indicates the overall stimulations and suppressions it has received from the antigen and other antibodies in the *HoF*. Higher the concentration of an antibody, larger the diverse set of antigens it can handle. In a sense, higher concentrations reflect the generic nature of an antibody. High concentration antibodies thus, have high performance. Concentration has thus been used to grade and select the best antibody from the *HoF*. A higher value of *Resource* indicates how consistently an antibody has been able to tackle antigens compared to the mutants and the *Ex-Champs* in the associated *HoF*. *Ex-Champs* that have a *Resource* value equal to or below 0 are definitely those who have been performing inconsistently. Thus, *Resource* has been used to decide the eviction of *Ex-Champs* from an *HoF*.

Using both concentration and *Resource* constitutes a win-win situation. Initially, when an antibody enters the *HoF*, irrespective of its concentration, its *Resource* value is high. This ensures a higher life time within the *HoF* and hence increases its chances of being selected as a *Champ*. Every time a *Champ* is selected and proves itself to be better than its *Mutant*, both its concentration and *Resource* increase. This results in an increase in its life time within the *HoF* as also its chances of becoming a *Champ* again.

If the value of concentration of any antibody in an *HoF* is high but its *Resource* value is low, it means that this antibody performed well initially and was re-selected as the *Champ* several times over the generations, but unfortunately in the further generations did not prove to be better than the *Mutant*. If a *Ex-Champ* is not se-



(a) (b)

Figure 2.19: (a) Concentration and (b) Resource values of the antibodies in the prominent *HoF* with ϵ set as 0.45 in *iAES-HoF* algorithm

lected for several generations, its concentration value would increase only marginally, as it receives the stimulation only from the antigen and not other antibodies in the *HoF*, while its *Resource* value would drop based on Eqn. 2.11. An antibody in an *HoF* with low values of both concentration and *Resource* thus clearly indicates that it was not selected as the *Champ* and that it has been lying in the *HoF* for a long period of time. Thus, a low value of *Resource*, irrespective of the value of the concentration, essentially indicates the high amount of time the antibody inhabited the *HoF* without being selected as the *Champ*.

It can be inferred that the *Resource* of an antibody can be used as the factor to decide whether or not the antibody should be evicted out of the *HoF*. *Resource* of an antibody increases as and when the antibody is selected as the *Champ*, thereby increasing its life span within the *HoF*. Had concentration been taken as the factor for such evictions then a fresh *Mutant* which has just beat the *Champ* would eventually get fewer chances of being selected as the *Champ*. In addition, one would need to define the maximum number of *Ex-Champs* (N) that could be accommodated within an *HoF*. Imagine a case when a *Mutant* $M_1(g)$ becomes the *Champ* in the g^{th} generation. Its concentration increases to $C_{M_1}(g)$ because of the stimulations from the antigen and all the current *Ex-Champs* in the associated *HoF*. If $M_1(g)$ were to perform well for the subsequent δ generations, and continue to reign as the *Champ*, its concentration would rise, to say $C'_{M_1}(g+\delta)$. If all the *Ex-Champs* in the associated *HoF* had earlier retained their position as a *Champ* for more than δ gen-

erations, their respective concentrations would be naturally more than $C'_{M_1}(g+\delta)$. If M_1 fails to retain its position as *Champ* after $g+\delta$, it would become an *Ex-Champ* and reside in the associated *HoF*. In successive generations, since its concentration C'_{M_1}, is lower than those of the others in the *HoF*, its chances of being re-selected as the *Champ* based on concentration is bound to remain very low. This would mean its concentration would never increase, making it a useless entity within the *HoF*. Subsequently, when the *HoF* is full, M_1 would become one of the prime candidates to be evicted. It is thus, evident that a concentration based eviction mechanism would never provide appropriate opportunities to the *Ex-Champs* within an *HoF* to participate in the selection process.

Unlike concentration, *Resource* values, are consumed or replenished based on 2.9 and 2.10 respectively. These values indicate how well the *Champ* or *Ex-Champ* performed over a window of time in the recent past. Higher values of *Resource* provide an indicator to its performance and indirectly suggest that it be retained for a longer amount of time within the *HoF*. *Resource* thus, acts as a measure of a dynamically varying life time of a *Champ* or *Ex-Champ* based on its performance. The decay of *Resource* within an *HoF* emulates the waning of the life time of an *Ex-Champ*. Eventually when the value of the *Resource* become zero or negative, it is apparent that the *Ex-Champ* has served its purpose and can now be removed from the *HoF*. It is for this reason that *Resource* is chosen as the parameter for eviction of an *Ex-Champ* from an *HoF*.

Figure 2.19a and Figure 2.19b respectively depict the *Concentration* and the *Resource* values of all the *Champs* that evolved over generations when ϵ was equal to 0.45 in real robot experiments for the task of PT-OA. In the Figure 2.19a, the steep rise in the *Concentration* of a *Champ* is due to the stimulations it received from both the antigen and the *Ex-Champs* in the associated *HoF*. The rise in concentration in case of an *Ex-champ* within an *HoF*, is not as drastic because it receives more of suppressions from the *Champ* than stimulations from the antigen. Suppressions and stimulations from the antigen and other antibodies thus tend to create a local idiotypic network [64] within each active region. In the Figure 2.19b, it can be observed that the *Resource* in case of *Champs* either remains constant due to almost equal consumption and replenishment (when there is no drastic increase in its fitness

over generations) or rises due to excessive replenishment (when there is an increase in its fitness over generations). On the contrary, the *Ex-Champs* within an *HoF* merely decay with time thereby ageing them, as seen in the Figure 2.19b The graphs for both *Resource* and concentration terminate at a point when the *Resource* becomes equal to zero indicating the eviction of the concerned antibody from the associated *HoF*. If the values of the positive magnifying constants α and β, used in calculating the values of *Concentration*, are increased (or decreased), then the respective values will only be magnified up (or down) without any changes in the pattern of the trend-lines seen in the graphs in the figures 2.19a and 2.19b. These constants do not affect the functioning of the algorithm.

2.7 Summary of the Chapter

This chapter aimed to achieve embodied lifelong learning, evolving controllers in an online and on-board manner with no previous information. The *iAES* algorithm evolved one controller for every active region, whereas the proposed *iAES-HoF* algorithm evolved a set of best controllers for each such active region by maintaining a *Hall of Fame (HoF)* for every active region.

While the use of the *iAES* algorithm resulted in one controller for every active region, the *iAES-HoF* algorithm evolved a set of the top performing controllers for each such active region by maintaining a *Hall of Fame (HoF)* for every such region.

The HoF comprises not only the current *Champ* but also the *Ex-Champs*. Instead of eliminating an *Ex-Champ*, the *iAES-HoF* algorithm adds it to the *HoF* of the relevant active region, thereby preserving the information learned by it. This generic set of controllers (or antibodies) within an *HoF*, including the *Champ* and *Ex-Champs*, is thus able to tackle most of the antigens within the associated active region. Antibodies within an *HoF* form an Idiotypic Network. Idiotypic Network in turn fortifies and increases the life times of the better performing antibodies while ensuring the eviction of the poorly performing and non-participating ones. The concepts of *Concentration* and *Resource* aided the enhancement re-selection and eviction mechanism of the antibodies within an *HoF*. In addition, these parameters regulated the number of antibodies in an *HoF* based on the dynamics of the sys-

tem, thereby circumventing the issue of deciding this number *a priori* consequently reducing a hyper-parameter.

The experiments also bring out the significance of the cross reactivity threshold, ϵ. This parameter is responsible for dividing the search space into several active regions such that a balance between the number of controllers and the active regions is maintained. Lower values of ϵ result in larger active regions, resulting in a higher number of antibodies in each *HoF*, encountering diverse antigens. The learning thus becomes slower. With higher values of ϵ, smaller active regions evolve with fewer antibodies in each *HoF*, encountering similar antigens. The switching of active regions was prominent in this case, which also resulted in slower learning.

The proposed work not only evolved the controllers in an online and on-board manner but also regulated their number dynamically, thereby achieving embodied lifelong learning. The work described in this chapter used conventional neuroevolution to evolve the neural controllers. The next chapter focuses on the second contribution of the thesis where a new mechanism to enhance the process of neuroevolution, has been described.

3

Enhancing Neuroevolution

Evolutionary algorithms (EA) deal with mimicking the Darwinian theory of natural evolution to solve real-world problems. In an EA, a population of individuals is created by using different techniques of selection, crossover, mutation, and reproduction. These operations are repeated over generations to eventually produce high performing or *fitter* solutions for the problem at hand [8]. Contrary to traditional techniques that work to refine a single solution, an EA does so on a population of such solutions. The convergence of such a population of solutions can be hastened by running the EA on parallel computing machines. EAs are mostly employed to solve optimization problems where the number of possible solutions is huge, and alternative heuristics-based solutions are computationally expensive. Some EA based solutions [34, 16] have been reported to even beat deep learning-based approaches in terms of both accuracy and time.

Conventional Artificial Neural Networks (ANN) are generally trained by adjusting the weights within, using gradient-based algorithms that greedily reduce errors using back-propagation. Neuroevolution [3] is another way of training an ANN wherein the weights are evolved using an EA. Such neuroevolutionary algorithms can also be used to evolve the architecture of the ANN, neural connections, the associated hyper-parameters, and even the learning algorithm. These are beyond the capability of traditional gradient-based algorithms. Of late, the techniques of neuroevolution have been used in tandem with deep learning algorithms [88] to obtain

state of the art results. An ANN can be trained using neuroevolution by altering the values in the weight matrices based on Gaussian mutation [14]. Generally, either all the weights within the matrices are mutated, or a randomly selected set is subjected to mutation. The ANN is then evaluated to check whether the mutation yielded the desired outcome or not. This process is continued until the required behaviors are evolved [97]. The performance of the ANN is measured based on a task-dependent *Objective* or *Fitness* functions, which helps calculate the *Fitness* values [132].

Not all mutations yield a favorable result. Some of these random mutations could prove to be beneficial in training the ANN, while others may cause adverse effects. Such pernicious effects delay the discovery of solutions, thereby increasing the time required for training. A mechanism to decide as to which weights need to be mutated should thus, be part and parcel of such learning algorithms.

In this work, a novel *mutational puissance* based strategy is proposed that can augment random mutation and help evolve ANNs with higher fitness values. A *mutational puissance* is allocated to every weight in an ANN. *Mutational puissance* provides an indication of the effect of past mutations of the associated weight on the performance of the ANN. It also gives an intuition of the mutational impact on the performance, if the respective weight is mutated.

The *puissances* associated with the weights are consumed and replenished based on the efficacy of the evolving ANN. If the performance of the ANN improves as a result of the mutation, then the *puissances* of the respective weights accumulate. Otherwise, these puissances dwindle, indicating an adverse effect on the performance. These mutational *puissances* guide the process of mutation consequently aiding the ANNs to evolve into better ones. To overcome the possibility of the ANN getting stuck in the local optima, random mutation is augmented along with the *puissance* based mutation. The proposed algorithm was used to evolve an ANN-based controller for a robot that eventually learned to execute the tasks of obstacle avoidance and phototaxis. The learning was performed online and on-board the robot.

3.1 Related Work

Few have reported techniques for moderating mutation in neuroevolution. Lehman et al. [78] proposed safe mutations for deep recurrent neural networks through output gradients. Their work focused on evolving deep neural networks without altering the behavior of solutions by using what they term as *safe mutation* operators. These operators are used on the weights to scale the degree of the mutation of each weight according to the sensitivity of the outputs. The operators facilitate the network to explore other areas in the search space without deviating from their functionality. The *safe mutational* operators evolve deep recurrent networks guided by gradient information without taking the performance into account.

Dahal and Zhan [22] have introduced a methodology to evolve the structures of Convolutional Neural Networks (CNN) by a series of recombination and mutational operators. Recombination was performed using the Highest Varying k-Features Recombination (HVk-FR) method. In their methodology, mutations are random and refer to aspects such as adding a new convolution layer, changing the size of a filter used, changing the activation function, etc. They have shown that their proposed method using combination, recombination, and mutation results in an ANN that has better accuracy.

Though mutation is used to overcome local optima issues, the use of random mutation may lead a search astray. In the work reported herein, a fairly controlled mutation mechanism is proposed that takes into account the performance of evolving the ANN. This work introduces a novel *puissance-based* mechanism to mutate the weights, which in turn can result in better-performing ANNs. The concept of a *mutational puissance* ensures that those weights that contribute adversely to the performance of the ANN are debarred from mutating, thus resulting in fitter ANNs.

3.2 Methodology

An evolutionary algorithm involves applying the operators of *Recombination*, *Mutation*, and *Selection* to evolve fitter individuals over generations. The evolved fitter individuals are selected and carried over to the next generation. In the purview of neuroevolution, various algorithms are proposed not just to evolve weight matrices

of the ANNs but also the architecture, associated hyperparameters, and even the associated learning algorithms. The proposed work focuses on evolving the connection weights of the ANN-based on a novel concept referred herein as a *Mutational Puissance*, ψ. Each weight in the weight matrices is associated with a *mutational puissance*, which changes as the ANN evolves. The purpose of this *puissance* is to moderate the mutation of the associated weight, thereby yielding fitter ANNs during the process of evolution.

The general idea of an evolutionary algorithm is, the operators of recombination or mutation are applied to the parent solution(s) to generate a child solution(s). If the child is fitter than the parent, the former replaces the latter. In this work, a simple fitness-based $(1+1)$ evolutionary strategy [14] is considered to evolve ANNs, where there is a parent ANN, and the child ANN is evolved by mutating the former.

3.2.1 Concept of Mutational Puissance

Nair et al. [91] have used the concept of a resource to model the natural waxing and waning of synthetic emotions. Every emotion has an associated emotional resource that is consumed and replenished based on the inputs to the systems. Penalties tend to consume them while rewards replenish such resources. These resources also decay with time irrespective of the rewards and penalties. The resource thus models the rise and fall of emotions. Godfrey et al. [49] have used a similar concept to control a population of mobile agents in a network in a decentralized manner. They have used the resource to model the concept of stigmergy and achieved unilateral backing off on the part of agents that clone. Jha et al. [63] describe the use of a potential that is similar to a resource that generates an innate drive within an agent that, in turn, aids it in deciding whether or not to switch from one behavior to another.

Mutational puissance (ψ), used herein, is based on similar grounds. It aids as a heuristic in the selecting weights that need to be mutated. Weights that have higher *mutational puissances* are the ones that are prone to be selected as candidates to be mutated. If the mutations result in an ANN exhibiting better performance, the *puissances* of the weights that were mutated are replenished; else, they are reduced. The *puissances* of the other weights that remain untouched, however, decay over time. This process is carried out for every generation to yield a better

set of weights eventually and, consequently, an ANN exhibiting better performance. The dynamics that govern the change in *mutational puissance*, ψ, are described in subsequent sections.

Consumption and Replenishment of ψ

The consumption of the ψ of the associated weights depends on how well the parent ANN performed against the evolving child ANNs. Consumption is proportional to the current value of the ψ and the difference in fitness values of the parent and the evolving child ANN.

If the performance of the parent is higher than that of the evolved child ANN, then the quantum of consumption of the associated *puissances* of the mutated weights increases proportionately.

The replenishment of ψ is proportional to the moving average of the differences in the fitness values of the parent and the child ANN over a window of generations. *Puissances,* of only those weights which are mutated, are consumed and replenished.

The value of ψ of the mutated weights at generation 'n+1' is given by,

$$\psi_{n+1} = \begin{cases} \psi_{max}, & \text{if } (\psi_n + \Delta\psi) > \psi_{max} \\ \psi_n + \Delta\psi, & \text{otherwise} \end{cases} \tag{3.1}$$

where the $\Delta\psi$ is the change in the ψ at generation n and is given by,

$$\Delta\psi = \psi_r - \psi_c \tag{3.2}$$

$$\psi_r = \mu_p * \mu_{ch} \tag{3.3}$$

$$\psi_c = \mu_{ch} * (\psi_n / \psi_{max}) \tag{3.4}$$

where ψ_r and ψ_c are the amount of ψ consumed and replenished respectively, μ_p is the moving average difference in the fitness of the parent of the previous generation and the current generation over a defined window of generations, μ_{ch} is the moving average difference in the fitness of the parent and the fitness of the child over a defined window of generations. This window is reset when a child ANN outperforms and replaces the parent. ψ_{max} is the maximum ψ (all these terms are

of generation n).

Decaying of Mutational Puissance

As stated earlier, for a given parent, only a certain number of weights are mutated. Hence, the *puissances* of only those weights which were mutated are consumed/replenished. The ones which do not participate in the mutation thus retain their original *puissances*. This causes an accumulation of ψ in those weights, which aided in the evolution of better ANNs. Accumulation of ψ may also be observed in weights whose mutations resulted in fitter ANNs in earlier generations but were overthrown by the current weights, which yielded ANNs with better fitness values. Intuitively, one may state that such weights with accumulated values have already been tuned to their current best values and hence should not be disturbed or mutated, unless required. Unfortunately, since the algorithm chooses weights with higher ψ, these weights tend to get selected again for mutation, causing instability in the fitness gradient. Under such conditions, the very objective of using mutational *puissance* to guide the evolution of the ANNs towards higher fitness values, is compromised. In order to avoid such a scenario and ensure that other weights also get selected for mutation, a decay mechanism is applied on the *puissances* of all such non-selected weights. A policy is formulated for the mutational *puissances*, ψ, to decay over generations. The ψ value of a non-mutated weight at the $(n+1)^{th}$ generation is given by,

$$\psi_{n+1} = \psi_n - \exp^{-\kappa} \tag{3.5}$$

where ψ_{n+1} and ψ_n are the mutational *puissance* in the $(n+1)^{th}$ and $(n)^{th}$ generations respectively, and κ is a constant, $(0 < \kappa \leq 1)$. In brief, the *puissances* of the mutated weights are updated as per the equation 3.4, while those of the non-mutated weights are updated as per the equation 3.5.

Algorithm 2: Mutational Puissance assisted Neuroevolutional Algorithm

```
initialise_puissance();
MaxGen ← Constant;
while MaxGen > 0 do
    MaxGen ← MaxGen − 1;
    if random() > P_re−eval then
        recover(Champion);
        F_Champion ← Run_Eval(Champion);
    else
        if random() < P_rdm_mutate then
            Challenger = Mutate_Random_Weights();
        else
            Challenger = Mutate_Puissance_Weights();
        recover(Challenger);
        F_Challenger ← Run_Eval(Challenger);
        if F_Challenger > F_Champion then
            Champion = Challenger;
            F_Champion ← F_Challenger;
            ψ_Champion ← ψ_Challenger;
            σ ← σ_min;
        else
            σ ← σ ∗ 2;
    Consume_Replenish_Puissance();
```

3.2.2 Proposed Mutational *Puissance* assisted Neuroevolutional Algorithm

A *puissance-based mutation* is incorporated in the (1+1) algorithm [14]. The (1+1) algorithm is the simplest version of the $(\mu + \lambda)$ algorithm, where a μ number of parents are mutated, resulting in a number of λ children. As it implies, (1+1) is a version of $(\mu + \lambda)$ where a single parent is mutated to produce a single child. In the (1+1) algorithm, the parent controller is termed as *Champion*. The weights of the *Champion* are mutated to produce the child, which is termed as the *Challenger*. Based on a random probability, either the *Champion* is reevaluated, and its fitness is updated, or the *Challenger* is generated and evaluated. If the fitness value of the *Challenger* is higher than that of the *Champion*, then the former replaces the latter. The extent of mutation of the *Champion* to generate a *Challenger* is governed by a parameter σ. This parameter is doubled whenever a *Challenger* fails to defeat

the *Champion*. When *Challenger* defeats the *Champion* the σ value is set to a predefined minimum value.

The proposed mutational *puissance* based algorithm is outlined in Algorithm 2. The *puissances* associated with the weights in the weight matrices are first initialized to the maximum value of the ψ, viz. ψ_{max}, and the maximum number of generations is also initialized to a value $MaxGen$. As per the original (1+1) algorithm, the *Champion* (i.e. the Parent ANN) or a *Challenger* (i.e., the Child ANN) which is evolved by mutation, is evaluated in every generation based on a random probability $P_{re-eval}$. If the current *Champion* is selected for evaluation its fitness, $F_{Champion}$ is updated. Otherwise, a *Challenger* is evolved using the proposed mutation algorithm. The *recover* function, just executes the *Champion* or *Challenger* ANN accordingly, without evaluating. This is to nullify any bias before the evaluation of the respective ANN-based controllers.

In this case, based on a random probability P_{rdm_mutate} the *Challenger* is evolved either by mutating random number of weights of the *Champion* (function $Mutate\_Random\_Weights$) or by mutating the weights of the *Champion* as per the ψ values associated with the weights (function $Mutate\_Puissance\_Weights$).

In the function *Mutate_Puissance_Weights*, the weights which have got higher *puissances* are made to mutate. A minimum of 25% of the total number of weights are empirically set to mutate in a weight matrix. Once the *Challenger* is evolved in either of the ways, it is then evaluated, and its fitness, $F_{Challenger}$, is determined. If the fitness of the *Challenger* is greater than that of the *Champion*, then the former is made to replace the latter as the new *Champion* and the value of σ, which is the extent of mutation, is set to a minimum value. If the *Challenger* fails to beat the *Champion*, then the σ value is doubled for the next generation. σ thus embeds somatic hyper-mutation [66] into the algorithm. After the re-evaluation of the *Champion* or the evolution of the *Challenger*, the *puissances* of all the weights are updated as per the equations 3.4 and 3.5.

The hybridizing of both random and ψ based mutation was done to ensure that those weights which have lower ψ values also get an opportunity in the evolutionary process. If this were not done, the weights having higher ψ would most likely hog the show and land the evolutionary process at a local optimum. Thus, a combination

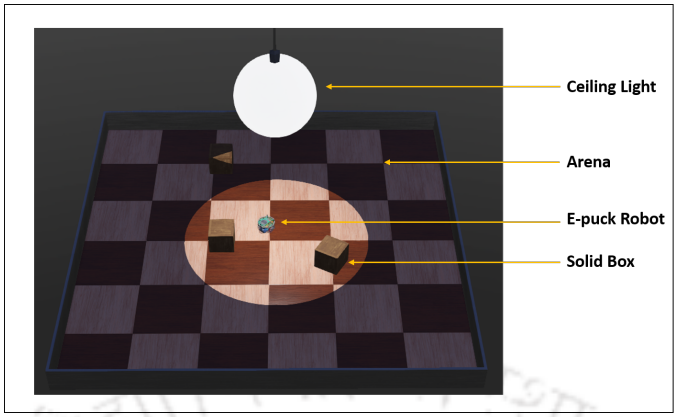


Figure 3.1: Arena in the Webots Simulator

of random mutations and ψ based mutations can be perceived to be a better option in evolving ANNs.

3.3 Experimental Setup

The proposed algorithm was used and tested in the domain of robotics; wherein neuroevolution was used to evolve ANN controllers. Experiments were conducted using the *Webots* simulator [87]. As depicted in figure 3.1, an arena of the dimension of 1.5*1.5m surrounded by walls 0.1m high was used in the experiments. The walls and the three squared solid boxes in the arena acted as obstacles for the simulated *e-puck* robot. A ceiling light, right in the middle of the arena, was used to light up the area, and the other parts of the arena were made to be dark.

The e-puck robot had eight IR distance sensors on-board, whose higher values indicated the presence of an obstacle, while the lower values indicated that the surroundings are obstacle-free. The robot was also armed with IR based light sensors on-board. The higher values of these sensors indicated the presence of high intensities of light, while lower values meant either the absence or lower intensities of light. The ANN being evolved was the controller of the robot, and the term controller and ANN are used interchangeably.

The algorithm was used to evolve two separate neuro controllers for two tasks, viz. obstacle avoidance and phototaxis. One set of experiments was conducted with the task of obstacle avoidance. In another set of experiments, the tasks were both obstacle avoidance and phototaxis, where the robot's goal was to evolve a controller

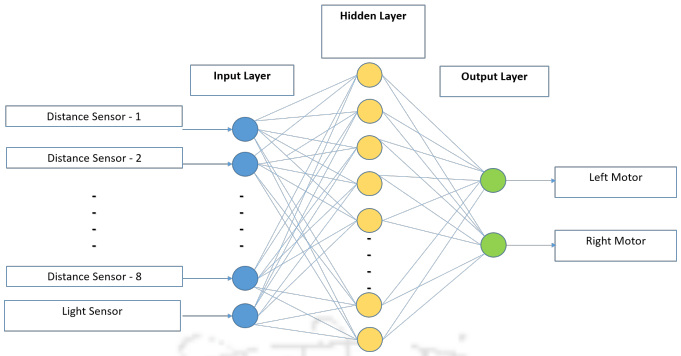


Figure 3.2: Structure of the Artificial Neural Network being evolved

that could avoid obstacles and also find its way towards the light source.

In order to evaluate the performance, the associated fitness value of each task was found. The fitness equation for obstacle avoidance is given as:

$$F_O = S_T * (1 - S_R) * (1 - d_{max}) \tag{3.6}$$

where S_T and S_R are translational and rotational speeds respectively, and d_{max} is the maximum distance value out of all the eight sensor readings obtained from the robot. The fitness equation for the phototaxis is given as:

$$F_R = F_O + L \tag{3.7}$$

where F_R is the fitness equation of the robot, F_O is the fitness of the obstacle avoidance task, and L is the value of the light sensor from the robot. It may be noted that the obstacle avoidance task is less complicated than phototaxis since the latter inherently includes the task of avoiding obstacles, too, during its movement towards the light source.

The structure of the ANN being evolved is shown in figure 3.2. The input layer corresponds to the sensors on-board the robot. Eight distance sensors, together with a light sensor, constitute a 9-node input layer. The hidden layer consists of 13 nodes, while the output layer has two nodes, corresponding to the speeds of the left and right motors, respectively. The values of the hyperparameters are as follows:

$\psi_{max} = 100$; $MaxGen$ was set to 400 for obstacle avoidance and 600 for phototaxis ; $P_{re-eval} = 0.3$; $P_{rdm\_mutate} = 0.3$; $\sigma_{min} = 0.01$ and $\kappa = 0.6$. The evaluation time in the *run_eval* function was set to thirteen seconds, i.e., the respective ANN controller was evaluated after it was run for thirteen seconds.

3.4 Results and Discussions

In this section, the results of the attempts to evolve the ANNs for the tasks of obstacle avoidance and phototaxis are discussed. Experiments were performed by varying the mutation strategies in the (1+1) algorithm as given below:

1. All Weights Mutated (AW-M): All the weights of all the weight matrices were mutated randomly.
2. Random Number of Weights Mutated (RMW-M): A random number of weights were mutated in the weight matrices.
3. ψ based mutation (ψ-M): The mutations of the weights were performed based solely on their ψ value.
4. Hybrid of Random and ψ based Mutation (RND+ψ-M): The mutations of weights were carried out using the proposed algorithm, both using random and ψ-based method as discussed earlier in algorithm 2.

In each of these experiments, the graphs of the fitness values of the evolving *Champions* vs. the Generations are plotted. The number of generations for the task of obstacle avoidance was set to 400, and that of the phototaxis along with obstacle avoidance was set to 600. The fitness values of the task phototaxis with obstacle avoidance are higher than the task of just obstacle avoidance because the fitness equation of phototaxis has got an extra term L, which is the value of the light sensor, as seen in the equation 3.7.

The graph in figure 3.3 depicts the variations in the fitness values of the *Champion* being evolved across generations in the AW-M strategy for the task of obstacle avoidance. As seen in the graph, the fitness curve falls to low values and, at times, to even zero. This indicates that a few mutations resulted in drastic falls in fitness

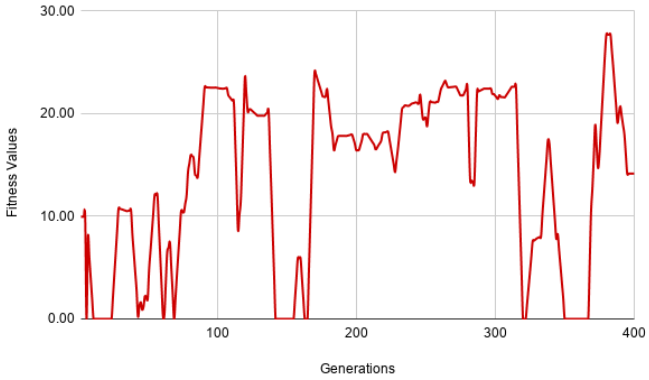


Figure 3.3: Variation in the *Champion* fitnesses of the AW-M version - Obstacle Avoidance

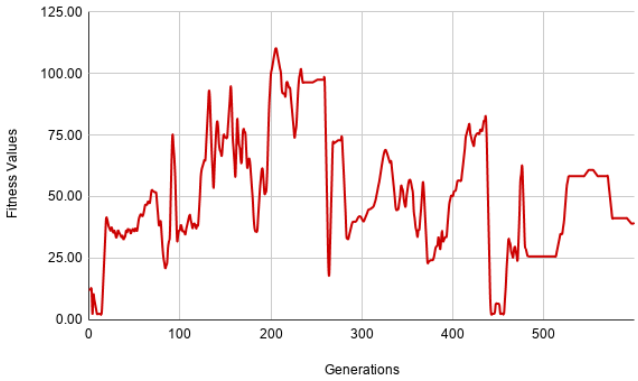


Figure 3.4: Variation in the *Champion* fitnesses of the AW-M version - Phototaxis

values. It inherently justifies that not all mutations contribute to improvement in performance. The graph in figure 3.4 depicts the variations of the *Champion* evolved across generations for the task phototaxis. Here too, the fitness values seem to rise and fall. The variations imply the complex nature of the task phototaxis, which makes it challenging to evolve a single ANN controller that can tackle such a task.

The figures 3.5 and 3.6 depict the fitness values of evolving *Champions* over generations for the tasks of obstacle avoidance and phototaxis, respectively, for the RMW-M strategy of mutation. In this version, as the mutations are completely random, there is nothing that guides the process of mutation. Hence, over the generations, there is hardly any improvement in the performance of the ANN controller. In the relatively complex task of phototaxis (shown in figure 3.6), too, the variations in the fitness values remain very random, suggesting that the learning has been

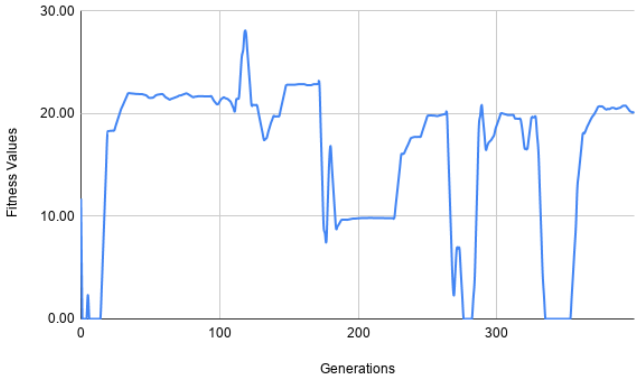


Figure 3.5: Variation in the *Champion* fitnesses of the RMW-M version - Obstacle Avoidance

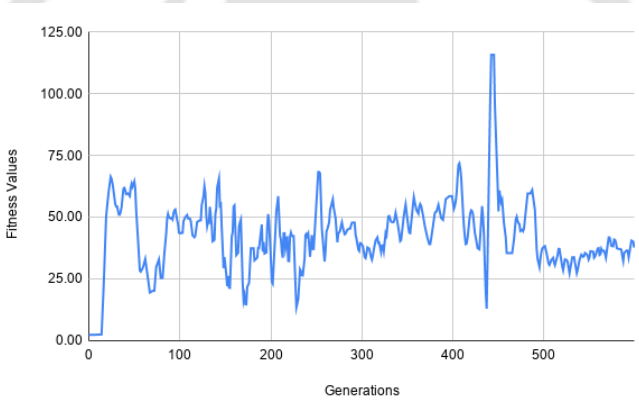


Figure 3.6: Variation in the *Champion* fitnesses of the RMW-M version - Phototaxis

unsatisfactory.

The figures 3.7 and 3.8 portray the fitness values of the evolving *Champions* over generations for the respective tasks of obstacle avoidance and phototaxis using the ψ-M strategy. This is a variation of the proposed algorithm 2, where the function *Mutate_Random_Weights* is skipped, and the weights are mutated solely on the basis of the values of ψ-M by the function *Mutate _Puissance_Weights*. Though this strategy performs better than the previous one for the simpler task of obstacle avoidance, as seen in figure 3.7, it does not seem to perform as much for the task of phototaxis. This suggests that the use of ψ based mutation does have an advantage over the previously discussed ones. However, it implies that merely following ψ-M based mutation limits the exploration and may not produce fitter ANNs, as seen in

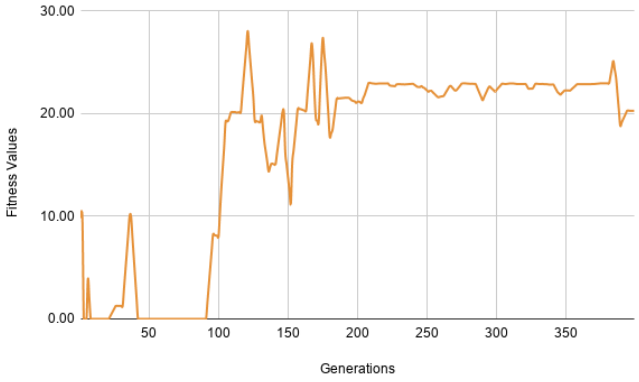


Figure 3.7: Variation in the *Champion* fitnesses of the ψ-M version - Obstacle Avoidance

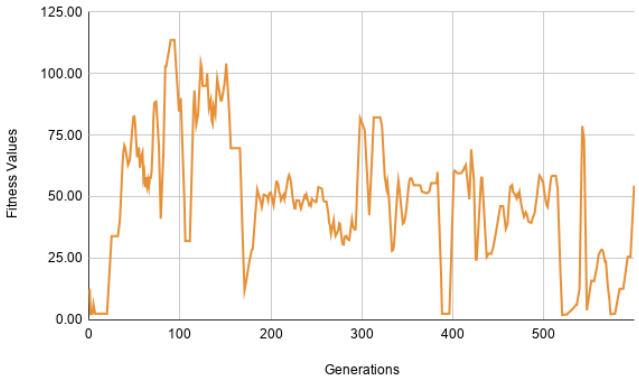


Figure 3.8: Variation in the *Champion* fitnesses of the ψ-M version - Phototaxis

figure 3.8.

The variation of the fitness values over the generations using the RND+ψ-M mutation strategy,(viz. the proposed mutational *puissance-based algorithm* 2), is depicted in the figures 3.9 and 3.10 for the tasks of obstacle avoidance and phototaxis. For obstacle avoidance, the graph shows a faster and stabler convergence than the ψ-M strategy, indicating that the hybrid version is more effective and provides a fair chance for all weights to participate in the evolution of the ANN. As for the task of phototaxis, the graph in figure 3.10 clearly outperforms its earlier counterparts by exhibiting a fairly steady increase in the fitness values indicating the effectiveness of the hybridized strategy. The mutational *puissance* seems to have guided the evolution of the ANNs towards better fitness values and not deviate if a fitter one is evolved. In contrast, the random mutations facilitate the much-required explo-

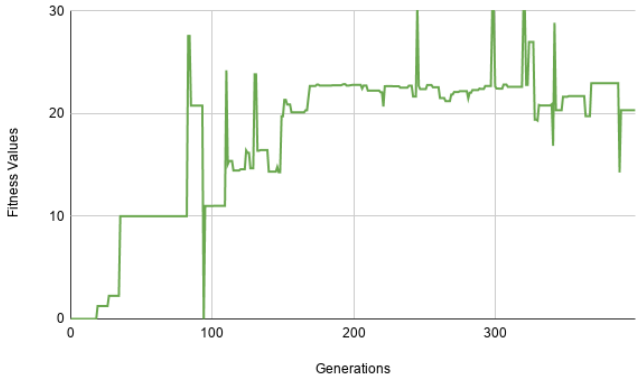


Figure 3.9: Variation in the *Champion* fitnesses of the RND+ψ-M version - Obstacle Avoidance

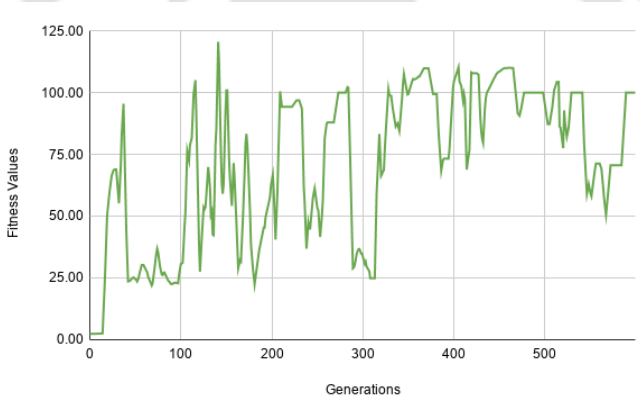


Figure 3.10: Variation in the *Champion* fitnesses of the RND+ψ-M version - Phototaxis

ration and ensure that the evolution does not get stuck in a local optimum. The proposed algorithm can thus be said to effectively balance random and *puissance* guided mutation strategies to evolve ANN controllers exhibiting better performances eventually.

Effect of Decay

The decay mechanism proposed in the algorithm avoids the accumulation of mutational *puissances* associated with the weights of the ANN. If these *puissances* get accumulated, it is highly possible that they misguide the evolution by forcing the associated weights to be selected more often for the mutation process. In order to test the efficacy of using such a decay, experiments are conducted using the proposed

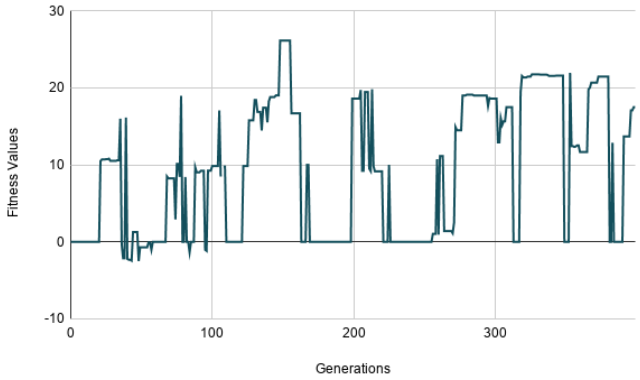


Figure 3.11: Variation in the *Champion* fitnesses of the RND+ψ-M version without the decaying of Puissance - Obstacle Avoidance

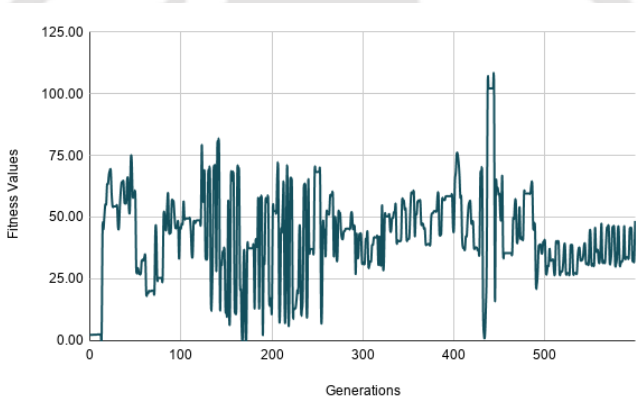


Figure 3.12: Variation in the *Champion* fitnesses of the RND+ψ-M version without the decaying of Puissance - Phototaxis

algorithm 2 but this time devoid of the decay mechanism. The graphs in figures 3.11 and 3.12 represent the variations in the fitness values of the *Champion* over generations for the tasks of obstacle avoidance and phototaxis respectively, when the decay of the associated mutational *puissances*, governed by the equation 3.5, was not incorporated in the RND+ψ-M version. From the graphs, it can be seen that the fitness variations exhibited are not as stable as those of their respective counterparts (graphs in figures 3.9 and 3.10), which used the decay mechanism. This is so because the removal of decay caused the accumulation of the *puissances* in the case of the non-mutated weights.

The weights with accumulated *puissances* were thus selected for mutation while evolving the *Challenger*. These weights were those that did not contribute to evolv-

ing better ANNs having higher fitness values. The removal of the decay mechanisms thus resulted in the creation of neuro controllers with poor performance, as seen in the graphs in figures 3.11 and 3.12. On the contrary, when the decay mechanism was incorporated, the graphs in figures 3.9 and 3.10, show the variations in fitness values over generations, to possess better and stabler learning or fitness curve. This justifies the need to incorporate a decay mechanism for mutational *puissance*.

3.5 Summary of the Chapter

This contribution described in this chapter, focused on a simple yet novel mechanism to enhance neuroevolution. In the neuroevolutionary techniques, the weights of an ANN are updated by the method of Gaussian mutation. During such a mutation process, either all or a random number of weights are updated. Such weight updations can result in deterioration of learning. The novel mutational strategy described herein mutated the weights of the ANN based on their respective *Mutational Puissance* values. These *Mutational Puissance* values are consumed or replenished based on whether the performance of the ANN deteriorated or improved. Mutation Puissances thus, acted as memories, capturing the contribution of the associated weights on the performance of the ANN. These values guided the mutation in the right direction, thereby improving the learning process of the ANN.

The results obtained from real experiments, indicate that just using Puissances could still lead the ANN towards a local optimum. A mix of random and mutational puissance based strategies seem to yield better results.

The contributions till now focused on learning the task in just only one environment. It may be noted that a robot could be transported to another similar or different environment. Learning in robots could be adversely affected when there is a change in the environment.

The next chapter discusses how the information learned in one environment can be transferred and used in another where a different task needs to be performed. Before formulating and implementing such a transfer of learned information in the open world of robotics, the next chapter focuses on a closed world problem to achieve such a transfer from one environment to another. The subsequent chapter, thus,

describes a new method to determine specific portions of an already learned ANN model and transfer them to another such ANN model which can be used in learning in the other environment. The source and target environments pertain to different datasets.

4

Neuronal Transfer Learning

Artificial Neural Networks (ANN) constitute one of the prominent and most effective supervised learning methodologies, where a mapping between a set of inputs and their corresponding outputs can be learned. These input-output pairs are most often in the form of a dataset. Part of the data within this set, is used to train the ANN while the remaining is used for testing. Learning is performed using the training set. The trained ANN (also referred to as a model) is then applied to the test data to crosscheck the accuracy and performance of the learning. Once trained, the ANN model can be used in conjunction with the dataset or data from the same domain. In other words, the trained ANN model often holds only for the same type of data on which it is tuned. The same training exercise and testing needs to be repeated as and when the dataset is changed. Rather than starting from scratch, researchers have tried to take some or all the layers from an already learned model of an ANN, to train on a new dataset. This method often leads to a reduction in the number of computations and consequently time. This reusing of already trained models to learn a new set of information is termed as *Transfer Learning* [100].

In transfer learning [81], either a few or all the layers of the trained source ANN (ANN_S) model are transferred to the target ANN (ANN_T) which is required to learn from a different dataset. The transfer of layers refers to transferring the associated weights of these layers to the target ANN. After transfer of the layers to the ANN_T, the corresponding weights are changed as per the learning algorithm,

along with the other ones. In other scenarios, the transferred weights are frozen in ANN_T and are not changed by the learning algorithm [117]. However, it is not always true that all these neurons in these transferred layers are beneficial in the training process. In fact, the non-beneficial neurons could, hamper and retard the learning process or even lead the search to a local optimum, culminating in a condition referred to as *negative transfer*[123]. Such adverse effects caused by certain neurons also increase the time required (or the epochs) to train the ANN_T.

It is thus, imperative to devise a manner in which one can identify neurons in the various layers that can prove to be beneficial, when transferred. This naturally would not only accelerate the training process but also culminate in a better and more accurate model.

This chapter presents an immuno-inspired strategy to identify neurons within an ANN_S which can ameliorate learning when transferred to the ANN_T. For clarity, such neurons have been termed as *Hot* neurons while the remaining ones are referred to as *Cold* neurons. Neurons within a specific layer are looked upon as a population and compete with one another during the learning process. The internal competition within a population of neurons in a layer, leads to the formation of a kind of *Local Idiotypic network* as described by Jha et al. [65] which in turn is similar to an Idiotypic Network [61]. The winning neuron which is the one with lowest loss, is stimulated by the rest of the neurons in the population causing an increase in its associated *concentration* or *temperature*. The winning neuron, in turn, also suppresses the other neurons in the population by decreasing their respective *concentration* or *temperature*. Over time the neurons that have been stimulated more often get hotter (increased temperature) while the others tend to become colder (decreased temperature). The temperature of a neuron thus, acts an indicator of its performance in terms of the losses it has suffered as compared to the others. The *hotter* neurons when transferred to the ANN_T show enhanced performance and low convergence times. Results obtained by performing such transfers while learning truth tables of logic gates and also Devanagari, Japanese scripts and English numerals throw insights into the effectiveness of this immuno-inspired Transfer Learning methodology.

The contributions of the work presented herein can be summarised as below:

- An immuno-inspired paradigm that can aid in identifying specific or *Hot* neurons in each of the layers within the source ANN that can be transferred to the target ANN, so as to enhance its accuracy and reduce convergence time.

- A heuristic that can convey, as to when to opt for this technique over conventional full layer-wise transfer methods.

4.1 Background

4.1.1 Transfer Learning

Of late, Transfer Learning (TL) has received more attention among the machine learning community who use deep learning. [100, 81, 127] report various categories of TL and their associated enhancements. In [118], Tan et al. describe TL where layers within deep neural networks are transferred. Fine-tuning an already trained model is a common technique followed by many researchers. Long et al.[80] present a method where only the last few layers of a deep network are fine-tuned. Kornblith et al. [71] explore various cases of fine-tuning in the transferred networks. Apart from tuning, it is also essential to identify the specific layers of an ANN that are to be transferred. Yosinski et al. in [133] study the transfer of initial, mid, and last layers of the deep networks. According to Yosinski et al., the information learned by the initial layers is more generic. Deeper into the network, the information becomes more specific to the data on which the network is being trained. Guo et al. [54], propose a strategy to select the layers which are to be fine-tuned. Based on a policy network, they decide whether the input is to be passed through a set of fine-tuned or pre-trained layers. Wei et al. [126], propose a methodology for TL based on past experiences. They initially, learn and optimize a reflection function which encrypts the transfer learning skills from past experiences. This function then aids in deciding the layers which need to be transferred. Rosenstein et al. [106] discuss on the question of whether to or not to, effect such a transfer. They apply the Hierarchical Naive Bayes method and conclude that the TL works best when the source and the target data are similar and not so well in the case of dissimilar data.

4.1.2 The Idiotypic Network

In the Biological Immune system, the non-self entities (termed as *Antigens*) are identified and eventually removed from the body. For eliminating such antigens, the immune cells secrete proteins called as *Antibodies* which are responsible for the tagging of antigens for their removal. Different antibodies recognize different types of antigens. The various antibodies can recognize each other as *self*. In his Nobel Prize winning work, Jerne [61] postulated that antibodies form a network among themselves which he termed as the *Idiotypic network* (IN). This IN is formed irrespective of the presence of an antigen. Whenever an antibody detects an antigen or other antibodies, the former antibody is stimulated by the latter; else, it is suppressed. These stimulations and suppressions result in an increase or decrease in the concentrations of the respective antibodies. INs have been used in a plethora of applications including Robotics and machine learning [109, 128, 58, 110, 111]. Jha et al. [65] have proposed the formation of Localized Idiotypic Networks (LIN), where various INs form in the vicinity of a set of antibodies at a node in computer network. They have described a novel architecture to emulate the Biological equivalent of INs. Bersini and Varela in [12] have proposed a methodology that hybridizes between the biological lessons and the engineering needs. They emphasize how the biological realities can assist the problem-solving in artificial neural networks and genetic algorithms.

In this work, concept of a IN is applied to identify the *Hot* neurons in a source neural network so that they can be transferred to the target neural network. Rather than identifying whole layers of neurons that need to be transferred and re-tuned, this chapter presents a method to identify the specific neurons within such layers, which when transferred can yield better performance of the target ANN.

4.2 Methodology

In conventional TL techniques (also termed as network transfer strategies), individual layers of the source ANN (ANN_S) are transferred to the target ANN (ANN_T). The ANN_S is the one initially trained on a certain dataset, D_S, while The ANN_T is the one that has to be trained using a different dataset, D_T. Transfer of a layer

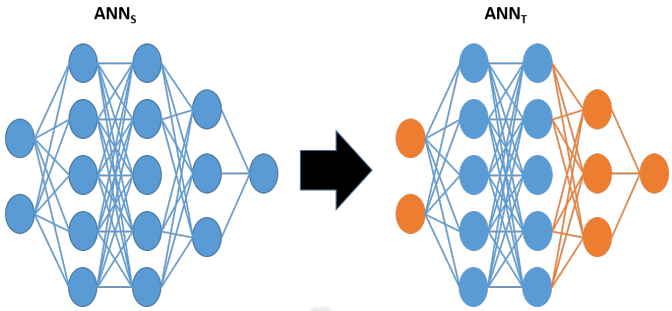


Figure 4.1: Two layers from ANN_S transferred to ANN_T

means that the associated weights of the neurons in those layers are transferred to the ANN_T. The figure 4.1 shows the transfer of two hidden layers, viz. the 2nd and 3rd, from an ANN_S to an ANN_T. All the weights in the ANN_T, except those corresponding to the transferred layers, are initialized randomly. During the training process of the ANN_T, the weights associated with the transferred neurons remain frozen (unchanged), while all the others change as per the back-propagation rule.

4.2.1 Immuno-inspired Idiotypic network based Transfer

This work proposes an Idiotypic Network (IN) inspired transfer of neurons from ANN_S to ANN_T, where only some of the neurons in the layers are transferred. As explained earlier in section 4.1, in an IN, the antibodies form a network by stimulating or suppressing the others based on conditions. This work metaphorizes the neurons as antibodies that form an Idiotypic-like network. The neurons in each layer constitute a population and form an Idiotypic Network which is local to that layer. If the number of neurons in a layer is large, then these are divided into populations of neurons, with an IN formed within each population.

Every neuron in the ANN_S is associated with a *Temperature* which is akin to the *Concentration* [111] of an antibody in the IN. At the end of an epoch in the training phase, the neuron which has lowest loss is stimulated by the others within its population. Likewise, this neuron suppresses the others. These suppressions and stimulations cause a neuron's temperature to increase (or decrease) with stimulations (or suppressions) much like in the biological Idiotypic Network. Thus, at the end of the training, some neurons in every layer will have higher temperatures than the others in their respective populations. The weights of these high

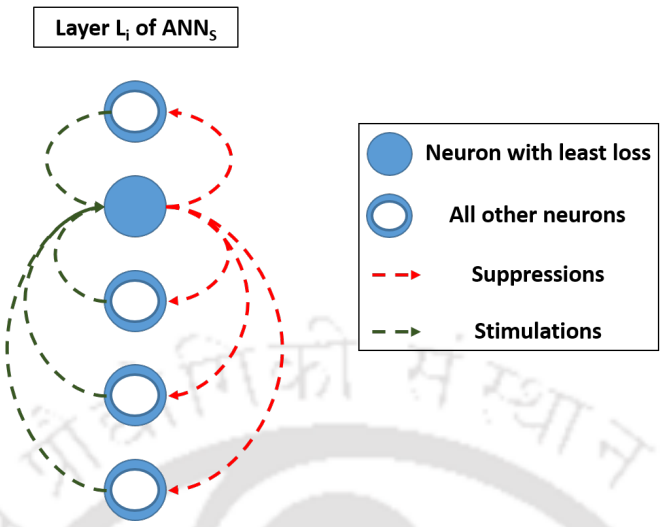


Figure 4.2: Stimulations and Suppressions of the neurons in a population in ANN_S

temperature, *Hot* neurons, are then transferred to the ANN_T and frozen so that back-propagation does not have any effect on them. Thus, instead of transferring the whole layer to ANN_T, only the *Hot* neurons are transferred. The weights of the other non-transferred neurons in the ANN_T are now re-tuned to achieve convergence and learning.

Ascertaining *Hot* Neurons in ANN_T

At every epoch, a neuron is stimulated if that neuron has the lowest loss in that population; otherwise, it is suppressed. This mechanism is shown in fig. 4.2. These suppressions and stimulations result in a change in the *temperature* associated to the neurons. To model the change in the temperature of the neurons, this work uses a variant of Farmer's equation [35].

Let the set L contain all the layers in ANN_S and $L^i \in L$ be the i^{th} layer in L. Let N^i be the set of neurons in this layer L^i, with n number of neurons in N^i. Every neuron $N^i_k \in N^i$, is associated with a *temperature*, Θ^i_k. These n neurons constitute a population which form an *Idiotypic Network* local to the layer L^i. In each epoch, let N^i_j be the neuron with the lowest loss in this population. The amount of stimulation received by N^i_j from all other neurons in this population is given by:

$$\tau_j^i = (1 - |\Psi_j^i|) * \omega_j^i \tag{4.1}$$
$$\omega_j^i = \sum_{k=1}^{n} (\beta * \Theta_k^i) \quad \text{where } k \neq j$$

where τ_j^i is the stimulation received by the neuron N_j^i, Ψ_j^i is the net loss at that neuron ($|\Psi_j^i|$ being the absolute value), β is a positive constant ($0 > \beta \geq 1$) to scale down the temperature values, ω_j^i is the sum of the temperatures of all other neurons except N_j^i and n is the number of neurons in that population. For every epoch, the suppression received by all other neurons in the population is given by:

$$\iota_l^i = \beta * (|\Psi_l^i|) * (\Theta_l^i / \Omega_l^i) \quad \text{where } \forall l \in N^i \text{ and } l \neq j \tag{4.2}$$
$$\Omega_l^i = \sum_{k=1}^{n} \Theta_k^i$$

where ι_l^i is the suppression received by the neuron N_l^i, Ψ_l^i is the net loss at that neuron ($|\Psi_l^i|$ being the absolute value) and β is a positive constant ($0 > \beta \geq 1$) to scale down the temperature values and Ω_l^i is the sum of the temperatures of all the neurons. The changes in *temperature* of the neuron with lowest loss, N_j^i, and that of all the other neurons are governed by the following equations:

$$\Theta_j^i = \Theta_j^i + \tau_j^i \tag{4.3}$$

$$\Theta_l^i = \Theta_l^i - \iota_l^i \quad \text{where } \forall l \in N^i \text{ and } l \neq j \tag{4.4}$$

Algorithm 3 portrays the method of identification of the *Hot* neurons in the training process of ANN_S. The Θ values of all the neurons are initialized before the training process begins. In every epoch, both the forward and the backward propagations are performed in the conventional manner and the losses for every neuron in each epoch are recorded. For every layer, the neuron which has incurred

Algorithm 3: Algorithm to ascertain *Hot* Neurons

```
Initialise Θ values
L^i ← set of layers from which neurons will be selected
for every epoch do
    forward_prop()
    backward_prop()
    for every L^i ∈ L do
        N^i ← Set of all the neurons in L^i
        j ← Index of the Neuron with lowest loss
        Calculate τ^i_j and Stimulate N^i_j
        for every N^i_l ∈ N^i where l ≠ j do
            Calculate ι^i_l and Suppress N^i_l
        Update Θ values of all neurons
for every L^i ∈ L do
    Determine ϑ^i
```

the lowest loss is stimulated by the rest of the neurons in this layer. The stimulations received by this neuron are calculated based on τ_j^i, Ψ_j^i and ω_j^i as per the equation 4.1. All other neurons in this layer are then suppressed by this neuron using ι_l^i and Ω_l^i as per the equation 4.2. The value of Θ associated to the neuron N_j^i and also all other neurons are updated using the equations 4.3 and 4.4, respectively.

Once the training is complete, the set ϑ^i is determined, comprising the top best *Hot* neurons within the layer i, with higher Θ values.

Each of the *Hot* neurons in this set has its respective associated set of neuronal weights. These weights are the ones that are transferred to the ANN_T in the TL process, so as to reduce the overall loss of training, consequently increasing the efficacy of the target model.

Transferring of *Hot* Neurons to ANN_T

After the *Hot* neurons in the ANN_S are ascertained during the training process, they are transferred to the ANN_T before training the latter. The weights associated with these transferred *Hot* neurons are frozen (remain unchanged) in the ANN_T while those of the others are initialized randomly and allowed to be re-tuned based on the propagation. Figure 4.3 depicts the transfer of the red coloured *Hot* neurons from ANN_S to ANN_T wherein they remain frozen (blue).

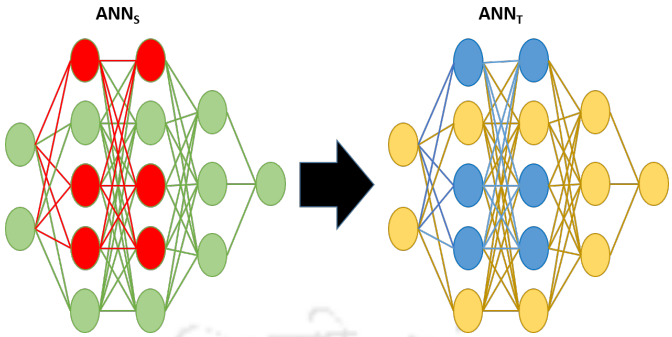


Figure 4.3: Idiotypic Network-based transfer of *Hot* Neurons (Red) from ANN_S to ANN_T where their weights remain frozen (blue)

After the transfer and freezing of the weights of the *Hot* neurons in the ANN_T, it is trained using the target dataset, D_T.

4.3 Experimental Setup

The proposed strategy of ascertaining the *Hot* neurons while training a source ANN and then transferring them to a target ANN was experimentally tried out in three different ways. The first set of experiments was targeted to a problem where ANN_S was trained to learn the XOR logic, and the corresponding *Hot* neurons were identified in the process. These *Hot* neurons were then transferred to two other target ANNs viz. ANN_{T_1} and ANN_{T_2} which were made to learn AND and OR logic, respectively. It may be noted here that, learning the target OR from source XOR logic is fairly easier, as the respective truth tables have higher similarity as compared to that of the target AND logic.

In the second set of experiments, the source ANN was a Convolutional Neural Network (CNN), representated as CNN_S, trained on the character dataset of Devanagari scripts [1]. The *Hot* neurons obtained during its training were then transferred to two other target CNNs viz. CNN_{T_1} and CNN_{T_2}. CNN_{T_1} was a CNN used to learn to classify handwritten digits in the MNIST dataset [76]. CNN_{T_2} was another CNN used to learn to classify ten rows of Japanese Hiragana in Kuzushiji-MNIST (KMNIST) dataset [19]. Here too, one may note that the source character dataset of the Devanagari scripts and the target datasets MNIST are fairly dissimilar. As for the target dataset KMNIST, this dissimilarity is very high.

In the third set of experiments, the source ANN used was a CNN trained on the MNIST dataset (CNN_S) whose *Hot* neuronal weights were used in a the target CNN_T comprising a CNN used to learn to classify the digits from the USPS dataset [59]. Since both the MNIST and USPS datasets contain English numerals 0 through 9, their contents are very similar.

To maintain pellucidity, the degree of transfer carried out is defined in the form of $\blacktriangleright^{\delta}$ where δ stands for the number of *Hot* neurons transferred (or the degree of transfer) from each population. $\delta = 0$ indicates no neurons were transferred, i.e., starting afresh using randomly initialized weights. $\delta = \lambda$ indicates that all neurons in the layer were transferred.

4.3.1 Experiment #1:Transfer of *Hot* neurons from a XOR ANN_S to OR and AND ANN_Ts

The ANN_S, which was trained to learn the XOR logic, had an input layer, two hidden layers comprising six neurons each, and an output layer. The neural network was implemented using the Numpy library. The sigmoid activation function was used in all the layers, and the temperatures Θ of all the neurons were initialized to 10. The learning rate was set to 0.1, and the network was trained for 10000 epochs. The value of β was set to 0.03. The targets, ANN_{T_1} and ANN_{T_2} used to learn AND and OR logic, respectively, had the same neural architecture as ANN_S.

In the case of the proposed Idiotypic network-based transfer, the *Hot* neurons from ANN_S were transferred from both the hidden layers to corresponding layers in ANN_{T_1} and ANN_{T_2}.

For comparison, the complete layer-wise transfer of neurons (where all the neurons of both the hidden layers of ANN_S were transferred to ANN_{T_1} and ANN_{T_2}), was also tried out separately. All the weights associated with the non-transferred neurons of ANN_{T_1} and ANN_{T_2} were initialized randomly, in both these cases, viz. the proposed Idiotypic network-based transfer and the complete layer-wise transfer. The weights associated with the transferred neurons, in both these cases were frozen during the training phases of ANN_{T_1} and ANN_{T_2}. The number of epochs used to train these target ANNs was fixed at 10000, just as the source ANN.

In order to get a better understanding, the transfer experiments were carried

out based on different conditions and their combinations. In this case, the experiments are conducted where the transfers were of the form: $\blacktriangleright^0$, $\blacktriangleright^3$, $\blacktriangleright^4$ and $\blacktriangleright^\lambda$. For each of these transfers, 15 trials of experiments were conducted.

4.3.2 Experiment #2: Transfer of *Hot* neurons from a CNN_S trained using Devanagari Character Dataset to those CNN_{T_1} and CNN_{T_2} trained using MNIST and KMNIST, respectively

The source ANN used was a CNN with three 5x5 convolutional layers and two fully-connected layers. The first convolutional layer consisted of 10 channels, followed by a ReLU activation. The second convolutional layer consisted of 20 channels, followed by a 2x2 max-pooling followed by a ReLU activation. The third convolutional layer consisted of 32 channels, followed by a dropout layer, 2x2 max-pooling, and ReLU activation. Two fully-connected layers with 512 and 200 neurons, respectively, followed these convolutional layers. Each of these fully connected layers was followed by the ReLU activation. A final softmax layer followed the fully connected layers. The number of output classes was 46. PyTorch framework was used to train all the CNNs viz. CNN_S, CNN_{T_1} and CNN_{T_2}.

The Devanagari character dataset consists of gray scale images each having a dimension of 32x32x1. The training and testing sets included 73600 and 18400 images, respectively. The hyperparameters used were as below: Learning rate = 0.01 ; Batch size during training = 64 ; Batch size during testing = 1000 ; Momentum = 0.5 ; Stochastic Gradient Descent (SGD) as the optimization method ; $\beta = 0.03$; Initial Θ value = 10.0.

Since each convolutional layer had a larger number of neurons as compared to the layers in the CNN_S, the neurons are divided into populations and evolved the Idiotypic Networks within each of them to reap the *Hot* neurons. The *Hot* neurons were identified and transferred from each of these populations within the convolutional layers to the convolutional layers in CNN_{T_1} and CNN_{T_2}.

The CNN_{T_1} for MNIST was a CNN that comprised two 5x5 convolutional layers and two fully-connected layers. The first convolutional layer consisted of 10 channels, followed by a 2x2 max-pooling and then by ReLU activation. The second convolutional layer consisted of 20 channels, followed by a dropout layer, 2x2 max-

pooling, and ReLU activation. Two fully-connected layers with 320 and 50 neurons, respectively, followed the convolutional layers. Each of these fully connected layers was followed by ReLU activation. A final softmax layer followed the fully connected layers. The number of output classes was 10. The CNN_{T_2} for KMNIST also had the same architecture as CNN_{T_1}.

Both the MNIST and KMNIST datasets consist of gray scale images having dimensions 28x28x1. 60000 images were used as the training set and 10000 for the test set. All hyperparameters were set to the same values as that of the CNN_S used to learn the Devanagari script.

For the Idiotypic Network based transfer, $\blacktriangleright^2$,$\blacktriangleright^3$ and $\blacktriangleright^4$ are followed, transferring respectively top 2, 3 and 4 *Hot* neurons from each of the population from the first two convolutional layers of CNN_S to the first two convolutional layers of CNN_{T_1} and CNN_{T_2}. For the complete layer-wise transfer $\blacktriangleright^\lambda$, all the neurons from the first two convolutional layers were transferred. For each of these transfers, 10 trials were conducted.

4.3.3 Experiment #3:Transfer of *Hot* neurons from CNN_S trained on MNIST to CNN_T to train on USPS

The architecture of CNN_S for MNIST in this case was same as that of the network used to train MNIST architecture mentioned in the subsection 4.3.2. The CNN_T for USPS was a CNN with two 5x5 convolutional layers and two fully-connected layers. The first convolutional layer consisted of 10 channels, followed by ReLU activation. The second convolutional layer consisted of 20 channels, followed by a dropout layer, 2x2 max-pooling, and ReLU activation. Both the fully connected layers were the same as the MNIST as elaborated in section 4.3.2, with the same values of the all the hyperparameters.

The USPS dataset consists of grayscale images, dimensions of each image being 16x16x1. There were 7291 and 2007 images in the training and test set, respectively. All other hyperparameters were the same as the MNIST in section 4.3.2. Here also, for the Idiotypic Network-based transfer,the following transfers are experimented: $\blacktriangleright^2$,$\blacktriangleright^3$,$\blacktriangleright^4$, $\blacktriangleright^\lambda$ Thus, the top 2, 3 and 4 *Hot* neurons, respectively, are transferred from each of the populations from the first two convolutional layers of CNN_S to the

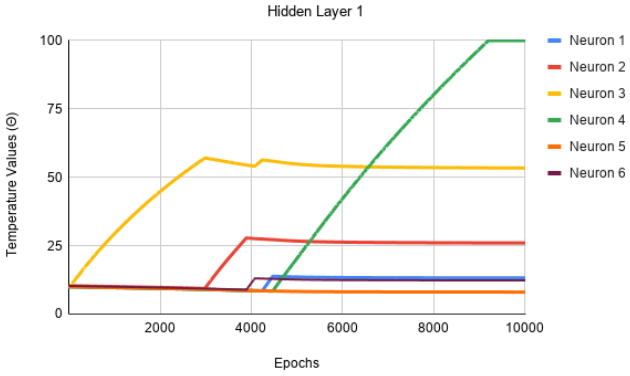


Figure 4.4: Variations of temperatures (Θ) of the neurons in the first hidden layer versus the epochs during the training of ANN_S to learn XOR logic

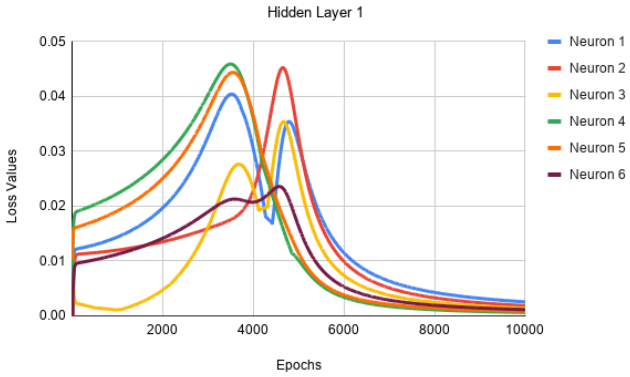


Figure 4.5: Variations of losses of the neurons in the first hidden layer versus the epochs during the training of ANN_S to learn XOR logic

first two convolutional layers of CNN_T. For each of these transfers, 10 trials were conducted.

4.4 Results and Discussions

This section presents the results of the proposed method of transferring *Hot* neurons for both the experiments described earlier.

4.4.1 Results from Experiment#1: XOR to AND and OR logic

The graphs in figures 4.4 and 4.5 show the variations of the temperature Θ and losses associated with each of the six neurons in the first hidden layer during the training of ANN_S to learn the XOR logic. It can be observed from these graphs, that when

the loss associated with a neuron as compared to others at a particular epoch, is lowest, it is stimulated. These stimulations tend to increase the temperature of the neuron with the lowest loss in that epoch. This neuron in turn suppresses the ones with higher losses thereby decreasing their respective temperatures. For instance, it can be observed in the figure 4.5 that initially the neuron-3 incurs the lowest loss till around the 3000^{th} epoch. This results in a rise in its associated Θ value as seen in the figure 4.4 due to the stimulations it receives from the other neurons during these epochs. On the contrary, the temperatures of all other neurons show a comparatively decreasing trend due to the suppressions received from the neuron-3. Between the epochs 3000 and 4000, since the neuron-2 incurs the lowest loss, its temperature increases while those of the others go down. Such trends cause the temperatures of the neurons having lower losses to increase over successive epochs and become *Hot* neurons.

Similar trends can be observed in the figures 4.6 and 4.7 for the second hidden layer. The ANN_S converged after 8000 epochs. Beyond this, the loss values for neurons in both the hidden layers become negligible and their corresponding temperatures attain different maximum values. Figure 4.8 and 4.9 depict the training losses over epochs when the transfers were carried out based on $\blacktriangleright^0$, $\blacktriangleright^3$, $\blacktriangleright^4$ and $\blacktriangleright^\lambda$ to learn AND and OR logic, respectively. In the case of AND, it can be observed that for $\blacktriangleright^3$ and $\blacktriangleright^4$, the losses of the corresponding target ANNs attained a near zero value much earlier than that of the $\blacktriangleright^0$. As for $\blacktriangleright^\lambda$, the corresponding target ANN converged to a higher loss indicating its poor performance. In the case of OR, the losses incurred in $\blacktriangleright^\lambda$ attained a near zero value slightly faster than that in the cases of $\blacktriangleright^3$ and $\blacktriangleright^4$, and significantly faster than the case $\blacktriangleright^0$.

It may be observed that in case of the XOR to AND transfer, the truth tables are fairly dissimilar, then a lower non-zero degree of transfer seems to perform better, whereas for the XOR to OR transfer, where the truth tables are similar, a higher degree of transfer performs better. Thus in the latter case of similarity, a full layer-wise transfer ($\blacktriangleright^\lambda$) is advised.

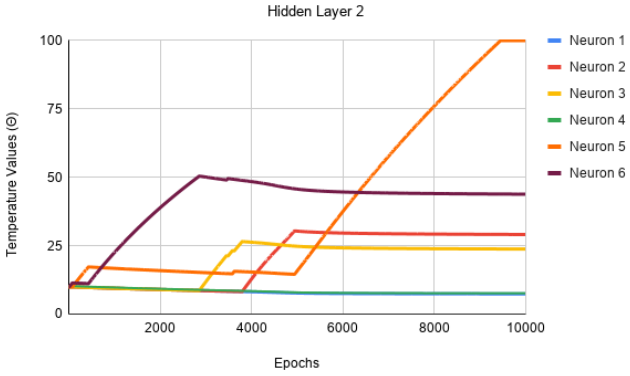


Figure 4.6: Variations of temperatures (Θ) of the neurons in the second hidden layer versus the epochs during the training of ANN_S to learn XOR logic

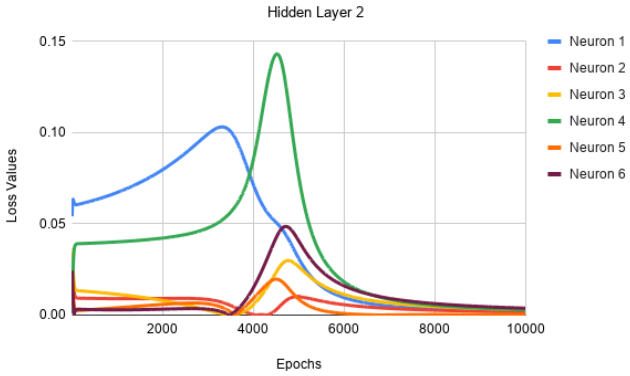


Figure 4.7: Variations of losses of the neurons in the second hidden layer versus the epochs during the training of ANN_S to learn XOR logic

4.4.2 Transferring *Hot* neurons from Devanagari Character dataset to learn MNIST and KMNIST data

Initially the CNN_S is trained as mentioned earlier, with the Devanagari character dataset. The *Hot* neurons were ascertained in every population of neurons in the first and second convolutional layers. These neurons were transferred to first and second convolutional layers of CNN_{T_1} and CNN_{T_2} to train using MNIST and KMNIST datasets, respectively. Table 4.1 shows the test accuracies for each of these cases over the epochs for the transfers: $\blacktriangleright^2$, $\blacktriangleright^3$, $\blacktriangleright^4$ and $\blacktriangleright^\lambda$. It can be observed that for both the MNIST and KMNIST datasets, the transfers $\blacktriangleright^2$, $\blacktriangleright^3$ and $\blacktriangleright^4$ outperformed $\blacktriangleright^\lambda$ in every epoch. Further, it may be noted that the transfers $\blacktriangleright^2$ and $\blacktriangleright^3$ performed better than $\blacktriangleright^4$. It can be observed from table 4.1 that for both MNIST and KMNIST

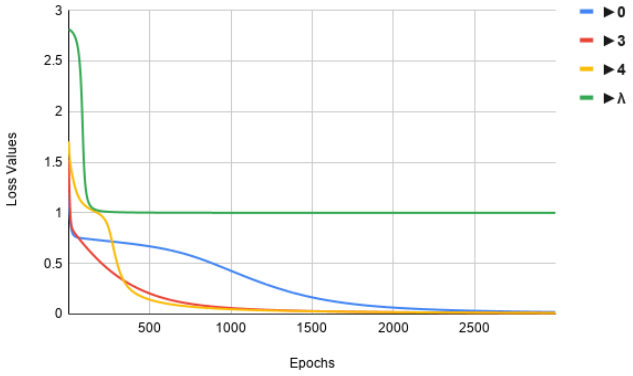


Figure 4.8: Variations of losses values over epochs during training of ANN_{T_1} to learn **AND** logic for $\blacktriangleright^0$, $\blacktriangleright^3$, $\blacktriangleright^4$ and $\blacktriangleright^\lambda$ transfers

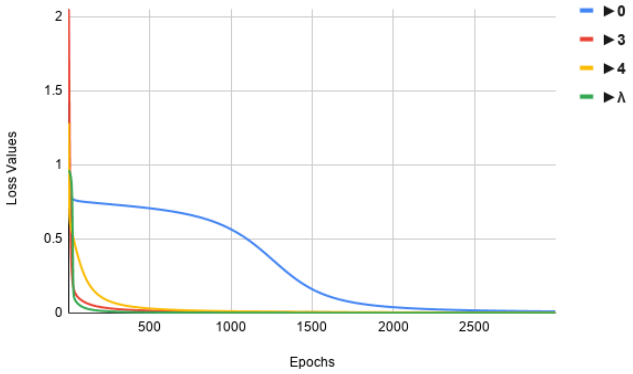


Figure 4.9: Variation of training loss over epochs during the training of ANN_{T_2} to learn **OR** Logic with $\blacktriangleright^0$, $\blacktriangleright^3$, $\blacktriangleright^4$ and $\blacktriangleright^\lambda$ transfer

datasets, that as the degree of transfers, δ, increases, the target CNN requires more epochs to reach the same accuracy values as those of the lower degree ones.

As discussed in section 4.4.1, here too since MNIST and KMNIST datasets are fairly dissimilar, the Idiotypic Network-based transfer of *Hot* neurons seems to perform better than the complete layer-wise transfer.

4.4.3 Transferring *Hot* neurons from MNIST dataset to USPS dataset

Here a source CNN was initially trained using MNIST dataset. After the training, the *Hot* neurons in each population of neurons in the first and second convolutional layers were ascertained were transferred to the first and second convolutional layers of CNN_T to learn the USPS data. Table 4.2 shows the test accuracies of the CNN_T

Table 4.1: Test accuracies of target CNN using MNIST and KMNIST after the transfer from source CNN trained using Devanagari Character dataset (values in bold indicate the best accuracies for each epoch)

Epoch	**Accuracies MNIST (in %)**				**Accuracies KMNIST (in %)**			
	$\blacktriangleright^2$	$\blacktriangleright^3$	$\blacktriangleright^4$	$\blacktriangleright^\lambda$	$\blacktriangleright^2$	$\blacktriangleright^3$	$\blacktriangleright^4$	$\blacktriangleright^\lambda$
0	**19.68**	16.89	7.41	14.11	10.11	7.10	9.29	**13.25**
1	93.31	**94.17**	93.21	93.20	67.41	68.41	**70.19**	69.54
2	95.56	**95.91**	94.61	94.14	74.34	73.12	**74.99**	70.77
3	96.44	**96.72**	95.70	94.43	**76.47**	75.80	76.29	72.47
4	97.09	**97.22**	96.22	94.44	**78.01**	77.81	78.22	74.46
5	97.16	**97.29**	96.41	95.05	**79.34**	78.77	78.70	76.24
6	**97.66**	**97.66**	96.86	95.26	79.91	**80.52**	79.59	76.69
7	**97.62**	**97.62**	96.88	95.31	80.78	**81.28**	79.63	75.99
8	**97.85**	97.79	97.31	95.59	**81.80**	81.58	80.00	75.85
9	**98.05**	97.74	97.08	95.57	**82.68**	82.61	80.69	77.60
10	**98.22**	97.99	97.42	95.70	83.22	**83.67**	80.61	76.90
11	**98.12**	98.00	97.29	95.52	82.91	**83.96**	82.24	76.95
12	**98.29**	98.03	97.70	95.76	**83.90**	83.67	81.74	78.07
13	**98.28**	98.16	97.68	95.99	84.43	**84.49**	82.14	77.47
14	**98.35**	98.26	97.63	95.67	83.97	**84.81**	82.56	77.94
15	**98.43**	98.12	97.80	96.22	84.20	**85.47**	82.06	78.85

over the the epochs for the transfers $\blacktriangleright^2$, $\blacktriangleright^3$, $\blacktriangleright^4$ and $\blacktriangleright^\lambda$. It can be observed from the table that the overall trend indicates for higher values of δ, higher performance is achieved in lesser epochs.

It maybe noted that the datasets MNIST and USPS are fairly similar and hence as observed in the results in the section 4.4.1, the $\blacktriangleright^\lambda$ performed better than $\blacktriangleright^2$ and $\blacktriangleright^3$.

From the results, it can be summarized that when the source and the target datasets are fairly dissimilar, the Idiotypic Network-based transfer using *Hot* neurons significantly outperforms the complete layer-wise method. When the datasets are similar, it can be seen that a complete layer-wise transfer performs better than the proposed method.

4.5 Summary of the Chapter

This chapter introduced a novel transfer learning method in neural networks, where instead of transferring complete layers of neurons, a set of pertinent or *Hot* neurons

Table 4.2: Test accuracies of target CNN using USPS after the transfer from source CNN trained using MNIST dataset (values in bold indicate the best accuracies for each epoch)

Epoch	**Accuracies USPS (in %)**			
	$\blacktriangleright^2$	$\blacktriangleright^3$	$\blacktriangleright^4$	$\blacktriangleright^\lambda$
0	10.41	12.51	**16.79**	16.54
1	56.55	62.88	77.03	**82.61**
2	76.38	79.57	83.31	**86.15**
3	81.96	82.86	86.95	**87.54**
4	82.96	84.65	87.99	**88.09**
5	84.90	86.15	88.89	**89.64**
6	87.10	87.39	**89.89**	89.84
7	87.49	87.89	90.13	**90.33**
8	88.14	88.99	90.48	**90.58**
9	88.34	89.39	90.28	**90.98**
10	89.14	89.59	**91.03**	90.48
11	89.04	90.58	**91.78**	91.28
12	90.13	90.53	**91.73**	91.28
13	90.08	90.28	91.18	**91.78**
14	90.43	91.18	**92.23**	91.48
15	90.58	91.23	**92.18**	91.83

was identified to be transferred.

An Immuno-Inspired algorithm proposed in this work aids in the identification of these *Hot* neurons from within the source ANN, which, when transferred to the target ANN, facilitates the faster convergence of the target ANN. The algorithm proposed used the concept of an Idiotypic Network to evolve the temperatures associated with the neurons within various layers of the source ANN. The results, on both shallow and deep neural networks, clearly indicate that the proposed neuronal transfer method outperforms the conventional layer based transfer in the cases when the source and target datasets are dissimilar. The chapter however, describes the use of the Immuno-Inspired transfer of neurons from source to target ANNs only in a closed world, i.e. learning from datasets made available. One cannot thus, be sure whether such a transfer mechanism will be beneficial in the robotics domain. It is important to verify whether this mechanism proves to be good, when a robot, whose controller has learned to adapt in one environment, is made to inhabit a different environment. The next chapter thus, describes how this Immuno-inspired neuronal transfer learning can be applied in an open world so as to achieve embodied lifelong

learning in robots.

5

Integrating Lifelong Learning and Transfer Learning Mechanisms

In the realm of evolutionary robotics, the controller of a robot is what actually drives it. This controller could, for instance be a Proportional–Integral–Derivative (PID) based one or one that uses a set of *if-else* rules, Bayesian networks or an Artificial Neural Network (ANN). Among these, the latter is widely used as the controller of the robot due to its smooth input to output transition. Conventional supervised learning controllers use ANNs constituting weights (or weight matrices), that are generally updated using backpropagation. However, in the case of neuroevolution, where the weights within these ANN based robot controllers are evolved using evolutionary techniques. These weights are updated by random Gaussian mutation. The quality of an ANN controller is determined by its fitness, which in turn indicates how well the robot performs the defined goal(s) using this controller.

In neuroevolutionary techniques, it is difficult for a single controller to learn to perform a task efficiently. Things deteriorate if the task is a complex one. Under such a condition, the evolutionary process suffers from the issues of bootstrap and deception and often ends up in local optima. Various methods have been introduced to overcome these issues which include incremental evolution, human-in-the loop, behavioral decomposition, etc. [46]. However, all these methods need some interference by an external entity and thus do not provide for an appropriate solution when

robot controller needs to evolve in an online and on-board manner. A recent work in [110], introduces an Immuno-Inspired methodology to evolve robot controllers even for complex tasks in an online and on-board manner. A hyper parameter termed the Cross Reactivity Threshold, denoted by ϵ divides the search space into various active regions on-the-fly. Instead of evolving a single controller, they describe the evolution of an ANN based controller for each such active regions within the search space. Thus, the robot is actually controlled by a set of controllers which have been evolved in an online and on-board manner. Using the sensor vector space, the robot determines which active region it is currently in, and uses the associated controller. In case, it has visited this region for the first time, it generates a new controller for this region using random weights and makes it learn them manner in which it needs to act in that region. They have shown that the use of this methodology aids in efficiently learning a complex task in the given environment.

However, the major drawback of this work [110] is that the it evolves just one controller per active region. The evolved controller is a mutated version of the former and replaces it if it performs better. Once the learned parent controller is replaced by the evolving (mutated) child controller, the learning achieved by the parent in the past, is lost. This phenomenon is akin to *catastrophic forgetting* [46] exhibited in neural controllers, where the already learned information is reset and thus, lost. The work reported in Chapter 2, proposes an enhanced Immuno-Inspired algorithm termed *iAES-HoF*, where a *Hall of Fame* (HoF) of controllers is maintained per active region rather than just a single one as in [110]. Rather than use a fixed population of controllers in an HoF, the number of controllers in each HoF is maintained dynamically based on the dynamics of the system. The technique ensures that the HoFs are populated by only the better performing controllers. The other non performing or unused ones are automatically evicted. Thus, many HoFs (one per active region), each caching a repertoire of the best evolved controllers are maintained for a given task and environment.

It may be noted this work pertains to a given environment. However, if a robot which has learned to inhabit an environment, E, needs to be translocated to another similar or dissimilar environment E', the overall evolutionary process may need to be started from scratch. This can be both time and energy consuming. Instead of

evolving from scratch, if the pertinent controllers evolved in E can be transferred to the robot situated in E' it can give it the right head start, rather than starting from scratch, thereby saving both time and energy. TL is known to work effectively when the source and target datasets (environments) are similar [119, 122]. If the source and target datasets (environments) are dissimilar, the source repertoire may not work well in the target leading to what is termed as *Negative Transfer* condition [134]. It is further challenging if the tasks that the source repertoire has learned is different from the target task that is to be learned. The work proposed in this chapter, describes a mechanism to identify the neurons to be transferred from a controller in the learned source repertoire to the target environment when the source and target environments are aiming to learn dissimilar tasks.

A transfer learning algorithm, termed as *NeuroEvotionary-Transfer (NeEvoT)* algorithm catering to embodied evolution identifies prominent neurons in each of ANN-based robot controllers within every HoF belonging to the source repertoire. We have associated a *Temperature* with every neuron within each of the controllers in this repertoire. While the source repertoire is being evolved, the *temperatures* of the neurons in the evolving controllers are changed proportionate to the gradients in both their respective fitness values and weights. Neurons with higher temperatures are tagged as *Hot* neurons. This source repertoire, along with *Hot* neurons, is transferred to the robot located in the target environment to accelerate its learning process. Simulations conducted using robots in the source and target environments indicate the efficacy of the transfer (NeEvoT) algorithm of repertoire in source and target environments for similar and dissimilar tasks.

The main contribution of this work is a transfer learning algorithm applicable for embodied neuroevolution in robots which can hasten the learning process in environments where the robots are required to learn similar or dissimilar tasks.

Subsequent sections provide TL and related work, the algorithm used for evolution of the repertoire of robot controllers, the *NeEvoT* algorithm, followed by experiments performed, their results, discussions and conclusions.

5.0.1 Transfer Learning

Transfer Learning (TL) is a mechanism where the knowledge acquired is transferred from a source domain to a target domain [82]. Such a transfer of knowledge is aimed to hasten the learning process in the target domain. TL is mostly successful in the cases where the source and target domains are similar [99].

In the realm of Evolutionary Algorithms (EA), various methods of TL have been proposed. Friess et al. [47] propose a method to learn an evolutionary search strategy that reflects the rough characteristics of a fitness landscape. This search strategy could eventually be transferred and adapted to other similar problems of interest. They focus on transferring the learned information to a target problem class that is similar to the source problem class. They have used benchmark problems to demonstrate effectiveness of their strategy.

Mouret and Maguire [90] have proposed a Multi-task MAP-Elites algorithm, which is an extension to MAP-Elites. Their algorithm solves numerous robotic tasks simultaneously by evaluating the fitness of individual tasks separately. They have used a high-performing solution of a robotic task to find solutions to other similar tasks. A task-selection operator is used to select similar tasks.

In [18], Chen et al. have described a TL based parallel evolutionary algorithmic framework for bi-level optimisation. In their framework, upper level variables parameterize a set of lower level problems and the latter is optimized. The performance of their algorithm was evaluated by experimental studies on various bi-level optimization benchmark problems. However, they state that the lower level problems must share some similarities with those of upper level ones during the optimization process.

Ma et al. in [83] have proposed a two-level TL algorithm to solve Multi-Task Optimisation (MTO) problems. They implement intra-task and inter-task learning at both lower and upper levels, across all dimensions based on transfer of decision variables. Correlation and similarity among the component tasks have been used improve the effectiveness of MTO. They categorically state that their transfer is effective only when the component tasks are similar.

In [21], Da et al. introduce an evolutionary framework to enable online learning

and exploit the similarities across optimization problems. Their work aims to achieve the algorithmic realization of the transfer optimization paradigm. They highlight that the similarities in the optimization problems may be less apparent on the surface and are revealed during the process of evolution. They have analyzed the efficacy their proposed work, both theoretically and practically, on a series of numerical examples.

In the Chapter 4, an Immuno-Inspired TL paradigm in a supervised learning setup for dissimilar datasets was proposed. The neurons, termed as *Hot* neurons, are identified from the source ANN only after completing the learning process. These *Hot* neurons are transferred to the target ANN, and their associated weights are frozen. The target ANN is then trained on a dataset dissimilar to the source dataset. Results obtained from experiments performed using both shallow and deep ANNs established the efficacy of their work in transfer learning, especially in the case of dissimilar source and target datasets. Their studies are restricted to learning using static datasets. In addition, since the source ANN needs to essentially complete learning of the associated dataset, the technique cannot be used on an *as is where is* basis in the realm of embodied evolution of robot controllers.

In all these works, except for the one in Chapter 4, transfer learning is effective only when the source and target tasks/environments are similar. Current state of the art transfer learning methods seem to have this as the main requirement. Literature survey indicates that no earnest attempt has been made to realize TL in evolutionary application domains where the sources and targets are dissimilar.

5.0.2 Evolving the repertoire of Robot Controllers - iAES-HoF Algorithm

Since our proposed work makes use of *iAES-HoF* algorithm to evolve robot controllers, a brief description of the same is provided herein. In this algorithm, the search space of a robot is dynamically divided into *Active Regions*. A *Hall of Fame* comprising robot controllers is evolved and maintained for each of these *Active Regions*. The set of all Halls of Fame constitutes a repertoire of robot controllers for a given environment. Two dynamically varying parameters viz. *Concentration* and *Resource* associated with of the controllers within an HoF control their re-selection

and eviction. The number of robot controllers within in each *HoF*, thus, varies based on the system dynamics. The incorporation of this algorithm in both the source and target environments hence results in lifelong learning in both the environments.

5.1 Methodology

This section elaborates the proposed *NeEvoT* algorithm for embodied evolution of robot controllers in two different environments. We denote the source and target environments as E_S and E_T, respectively. A robot controller constitutes an ANN, which uses an evolutionary strategy and in turn drives the robot. The fitness of this evolving controller is determined by a task-dependent fitness function.

5.1.1 *NeEvoT* Algorithm

For clarity we denote Re_S and Re_T as the repertoire of robot controllers being evolved by two distinct robots inhabiting the source and target environments E_S and E_T respectively. E_T is comparatively more a complex environment as compared to E_S. The Re_S of the robot in E_S is evolved continuously using the *iAES-HoF* algorithm as described in Chapter 2. In Re_S, a neuron within a controller is associated with a *Temperature* (Θ).

The ANNs used in conjunction with the TL algorithm described in Chapter 4, used supervised learning. At the end of every training epoch (or iteration), the value of loss was based on the predicted value. This loss was back-propagated to every layer of the ANN, and consequently, to every weight based on which the neuronal *temperatures* were updated. However, in the case of evolutionary learning in ANNs, the weights are randomly mutated using Gaussian mutation, rather than backpropagation. Since the concept of loss is absent in this case, there is a need of a new measure to determine the values of temperatures associated with the neurons.

Determining Hot Neurons in E_S

In this work, the combination of fitness values of the controllers being evolved and their respective weights are considered in determining the temperature of the neurons. Let the value of the fitness of a controller be f_g at generation g. The weights

Algorithm 4: Algorithm to update the θ of Neurons in Neuroevolution on controllers

```
initialize_θ();
MaxGen ← Constant;
while CurrGen < MaxGen do
    CurrGen ← CurrGen + 1;
    get_best_controller();
    mutate_controller();
    f ← evaluate_controller();
    for every L^i ∈ L do
        for every neuron do
            determine_θ();
            update_θ();
    update_Re_S();
```

of this controller are mutated in the next generation $g+1$ and its corresponding fitness $f_{(g+1)}$ is evaluated. If $f_{(g+1)}>f_g$, it indicates an improvement in the learning implying that the controller evolved at $(g+1)$ is better than its predecessor and the mutations have proved to be effective; else there is a degradation. Since, the change in weights as also fitness values govern the quality of learning over the generations, the temperature of each neuron i of a controller j is calculated using the equations below:

$$\theta_{new} = \Delta f * \Lambda \tag{5.1}$$

$$\Delta f = f_{g+1} - f_g$$

$$\Lambda = \sum_{k=1}^{n} (w[k]_{g+1} - w[k]_g)$$

$$\theta_{g+1} = \theta_g + \theta_{new} \tag{5.2}$$

where Δf denotes the difference in the fitness values of the $(g+1)^{th}$ and g^{th} generations, Λ is the sum of the differences in the weights at the $(g+1)^{th}$ and the g^{th} generations of a neuron. The temperatures of all the neurons in all the controllers are calculated for every generation. If the gradient of the change in the weights of a

neuron over subsequent generations is negative, it implies that the neuron has not contributed significantly to the learning process. As can be seen from the above equations, under such conditions the increase in temperature could be insignificant or negative. Thus, over generations, the temperature of a neuron will indicate its influence on the learning exercise. Higher values of the same indicate that the associated neuron is *hot* and has positively contributed in the learning, making it a potential candidate for a transfer to E_T.

Algorithm 4 portrays the method to determine the temperature, θ of a neuron. Initially, the values of θ are conferred the same positive constant values. In the function *get_best_controller*(), the best controller from the pertinent *HoF* at that instant of time is retrieved from the Re_S and mutated after which the fitness of the mutated controller is determined. The temperatures of all the neurons in the L layers that need to be transferred is then determined as per the equations 5.1 and 5.2. The top n hot neurons within the transferable layers of every controller are marked and the complete source repertoire Re_S is transferred to the target robot where it serves as its *external repertoire* (Re_{Ex}).

5.1.2 Learning in the Target Environment

In order to evolve the target repertoire Re_T, in the Target environment E_T, we have used the same *iAES-HoF* learning algorithm. The transferred repertoire from E_S to E_T is termed as an *External Repertoire* Re_{Ex} in E_T. At every generation, at the step *get_best_controller()*, the best controller is chosen either from the Re_{Ex} or Re_T based on a random probability, p_{ex}. Since the target robot now has two repertoires - one being its own and the other which is externally transferred - it needs to choose the best controller from one of these. Selection of the repertoire is done based on random probability. If it chooses a controller from its own internal repertoire, then it follows the procedure detailed in the *iAES-HoF* algorithm. Otherwise, it extracts the weights associated with the hot neurons of the best controller from the external repertoire, Re_{Ex}. A new controller with random weights is first generated. The weights of the hot neurons are then conferred to the appropriate links within this newly generated controller which forms the *Challenger*. If this challenger outperforms the current best controller from Re_T, then the former is added to Re_T; else

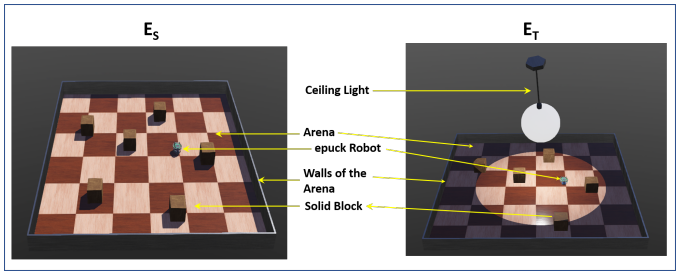


Figure 5.1: Source E_S and the Target E_T environments in the Webots simulator

it is discarded.

5.2 Experimental Setup

The experiments to test the proposed algorithm were carried out using the *Webots* simulator [87]. Snapshots of the source, E_S, and the target, E_T, environments within two separate instantiations of the Webots simulator are shown in the figure 5.1. Both E_S and E_T had the same arena dimensions of 1.5mX1.5m, surrounded by walls 0.1m high. In both the environments, the solid boxes and the walls acted as obstacles. As can be observed in 5.1, the placement of the obstacles is however, different. In addition, the target environment, E_T, has a light suspended from the ceiling at the centre of the arena. The simulated robot used was an *e-puck*, which has eight IR distance sensors and eight IR light sensors.

The robot in in E_S was made to learn the task of obstacle avoidance. The tasks to be learnt by the robot inhabiting E_T was complex. In addition to obstacle avoidance, this robot needed to learn the task of avoiding obstacle while performing phototaxis. The fitness functions used were the same as that in Chapter 2 (equations 3.6 and 3.7). It may be noted that though the task of obstacle avoidance is common to both the robots, ANN based controllers evolved by the robots in E_S and E_T are completely different. Therefore from the perspective of neuroevolution, the tasks maybe deemed to be dissimilar.

The architecture of the ANN being evolved in both the environments were same. ANNs with five input neurons corresponding to the four distance sensors and a light sensor on-board the robot, followed by a hidden layer with eight neurons and a two-neuron output layer were used in the experimentation. The hidden layer

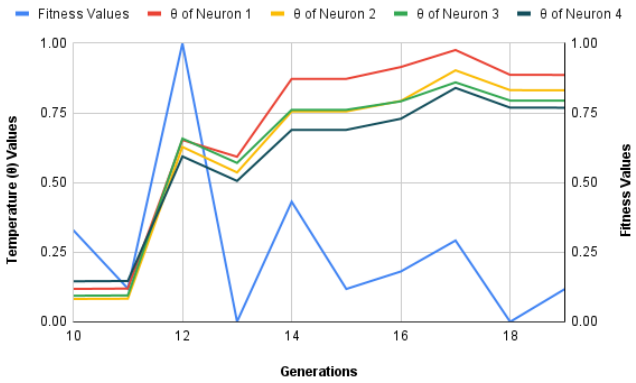


Figure 5.2: Variation in the θ values and the *Controller* fitness values over a short term of generations in E_S

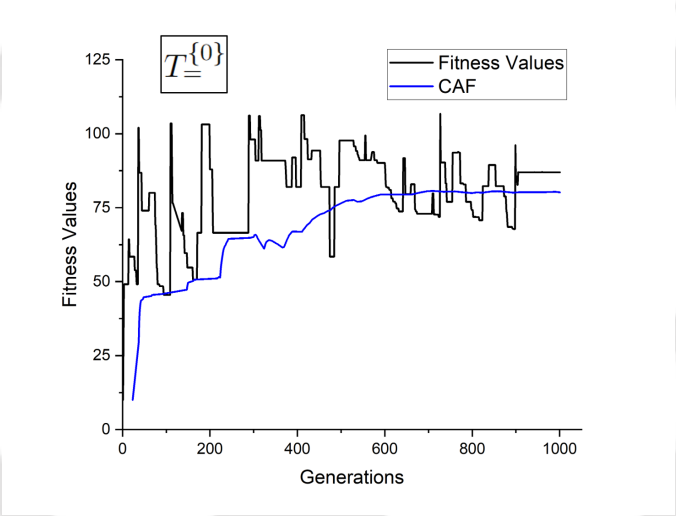


Figure 5.3: Variation in the *Controller* fitness values over generations in the case $T_{=}^{\{0\}}$ in E_T

within these ANNs formed the layered to be trsansferred. The hyperparameter values used in both E_S and E_T are as follows: $MaxGen$ = 1000 ; in E_T only: p_{ex} = 0.5.

5.3 Results and Discussions

Experiments were carried out as per the settings elaborated in the previous section 5.2. Two sets of experiments were performed. In the first set, denoted as $T_{=}$, both the environments, E_S and E_T, evolved controllers to carry out the same task of obstacle avoidance. The second set, designated $T_{\neq}$, comprised the case where E_S

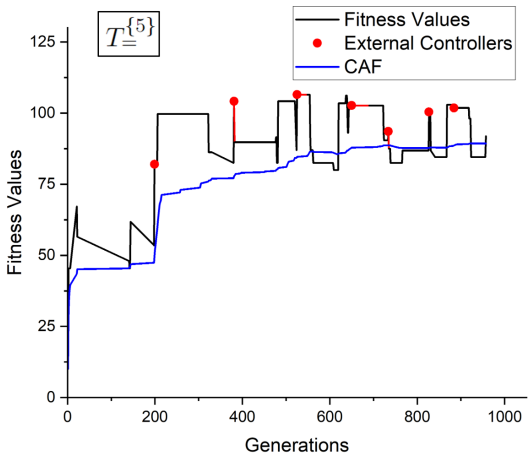


Figure 5.4: Variation in the *Controller* fitness values over generations in the case $T_{=}^{\{5\}}$ in E_T

evolves controllers for the task obstacle avoidance. The repertoire Re_S including the hot neurons, generated in this experiment, was then transferred to E_T so as to evolve Re_T for a task different from that learned in E_S. This new task was that of obstacle avoidance cum phototaxis, just as the one cited in Chapter 2. It may be noted that the robots in the source and target environments do not commence their learning at the same time. The robot in E_S, was made to have a head start so that it performed a substantial amount of learning before effecting a transfer of its repertoire to the other robot in E_T. The robot in E_T was thus, made to commence its learning only after it peer in E_S completed 500 generations. In each set of experiments, $T_{=}$ and $T_{\neq}$, the following runs were conducted:

1. *Case 1* - $T_{=}^{\{0\}}$ *or* $T_{\neq}^{\{0\}}$: Evolving the controller from scratch and utilizing transferrred repertoire

2. *Case 2* - $T_{=}^{\{5\}}$ *or* $T_{\neq}^{\{5\}}$: Transferring top 5 Hot Neurons from the hidden layer

3. *Case 2* - $T_{=}^{\{6\}}$ *or* $T_{\neq}^{\{6\}}$: Transferring top 6 Hot Neurons from the hidden layer

4. *Case 2* - $T_{=}^{\{7\}}$ *or* $T_{\neq}^{\{7\}}$: Transferring top 7 Hot Neurons from the hidden layer

5. *Case 2* - $T_{=}^{\{\lambda\}}$ *or* $T_{\neq}^{\{\lambda\}}$: Transferring all the neurons from the hidden layer, i.e., complete layer transfer

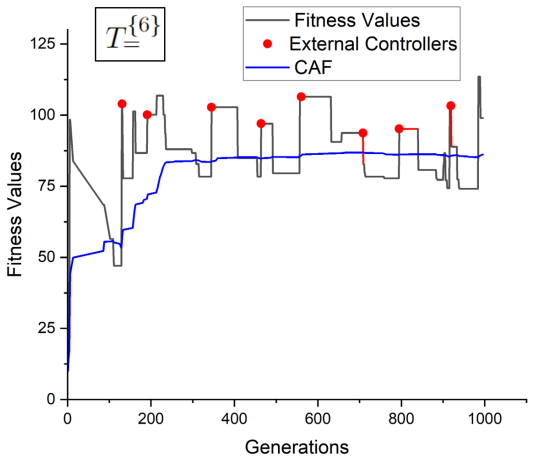


Figure 5.5: Variation in the *Controller* fitness values over generations in the case $T_{=}^{\{6\}}$ in E_T

Before elaborating on the results obtained from these experiments, it may only be apt to notice the significance of the variation of temperature, θ on the extent of learning, over the generations. Figure 5.2 depicts the variations of temperature and fitness (the learning performance of a controller) over some generations. As can be seen from this graph, initially from the 11th to 12th generation, the fitness increases sharply. This nature is followed by the temperatures of all the neurons. From the 12th 13th generation, as the fitness drops, there is a slight negative gradient in temperatures of all neurons. Subsequent portions of variation in fitness indicate clearly that whenever the fitness gradient is positive, it increases the current temperature of the neurons, proportionately. It can also be observed that from the 13th to the 14th generation, the fitness rises causing the corresponding temperatures to also increase. The temperature of *Neuron 1* can be seen to increase more sharply mainly because of higher difference between the weights of the current and earlier generations. This essentially means that the weights of this neuron were mutated to a higher extent, which in turn implies that *Neuron 1* seems to play a more important role in the learning process as compared to the other neurons. In brief, one may conclude that both fitness and the extent of mutation of weights, both play a vital role in governing the temperature of a neuron. The temperature of a neuron thus, aggregates both the fitness values and the extent of mutations performed over

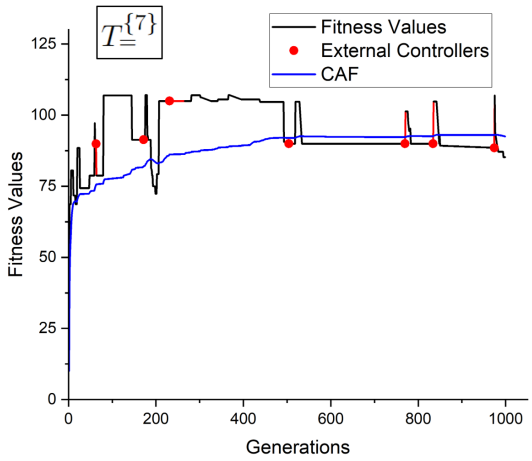


Figure 5.6: Variation in the *Controller* fitness values over generations in the case $T_{=}^{\{7\}}$ in E_T

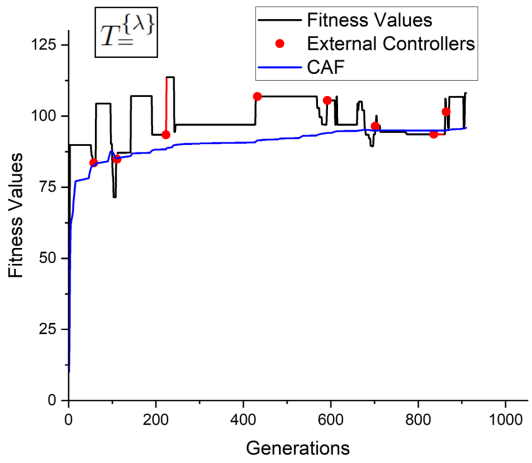


Figure 5.7: Variation in the *Controller* fitness values over generations in the case $T_{=}^{\{\lambda\}}$ in E_T

generations, thereby making it a fair indicator of a neuron's contribution to the learning process. Thus, transferring of the hotter neurons to the target can hasten the learning process at the target. Temperature of a neuron can act as a metric while identifying and transferring neurons to the target environment.

5.3.1 Evolving Similar Tasks $T_{=}$

The graphs in the figures 5.3-5.7 depict the variations in fitness values and the Cumulative Average Fitness (CAF) over generations for the transfer of the repertoire from E_S to E_T, where both the environments are evolving the same task of obstacle avoidance ($T_{=}$). Though the task remains the same, it may be noted that the environments are not similar. The controller thus, needs to adapt to E_T. To avoid clutter, all graphs for each of the transfer cases, have been depicted in different figures. The best controllers evolved from the shared external repertoire (External repertoire Re_{Ex}) are highlighted in the graphs (figures 5.4 - 5.7) using tiny red dots. Note that the figure 5.3 has no red dots, as it represents the case of no transfer,$T_{=}^{\{0\}}$. In the case of $T_{=}^{\{0\}}$ (figure 5.3), where there is no transfer of repertoire (i.e. the controller evolves from scratch), it can be observed that there are numerous rises and falls in the fitness values over generations and the learning curve is not smooth. In addition, the CAF curve does not drastically fall to zero due to the underlying *iAES-HoF* algorithm and the presence of *HoFs*.

In the cases $T_{=}^{\{5\}}$ and $T_{=}^{\{6\}}$ (figures 5.4 and 5.5), where the top 5 and 6 neurons are transferred, respectively, the graphs seem to improve in steadiness with higher CAF values as compared to that in case of $T_{=}^{\{0\}}$, indicating a marked improvement in performance. In the cases $T_{=}^{\{7\}}$ and $T_{=}^{\{\lambda\}}$ (figures 5.6 and 5.7), the curves seem to fluctuate lesser and end up with higher CAF values than the previous ones. One may therefore conclude that when tasks are same, higher order transfers (i.e. when more number of hot neurons from a layer, are transferred) as in the top 7 (figure 5.6), and complete layer transfer (all neurons in that layer as in figure 5.7) exhibit superior performances.

In lower order transfers, like $T_{=}^{\{5\}}$ and $T_{=}^{\{6\}}$, only a few top hot neurons are transferred while the weights of the others initialized to random values at the target end. This causes the controller to forget a portion of the already learned task making it toil again in E_T to regain and relearn the same task. It is because of this that we observe greater fluctuations in the fitness curves in figures 5.9 and 5.10 as compared to higher order transfers. In higher order transfers, $T_{=}^{\{7\}}$ and $T_{=}^{\{\lambda\}}$, since, the corrections to be made to the weights is far lesser, the curve exhibits

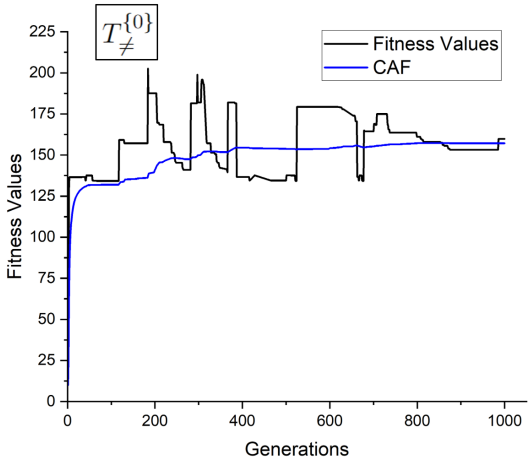


Figure 5.8: Variation in the *Controller* fitness values over generations in the case $T_{\neq}^{\{0\}}$ in E_T

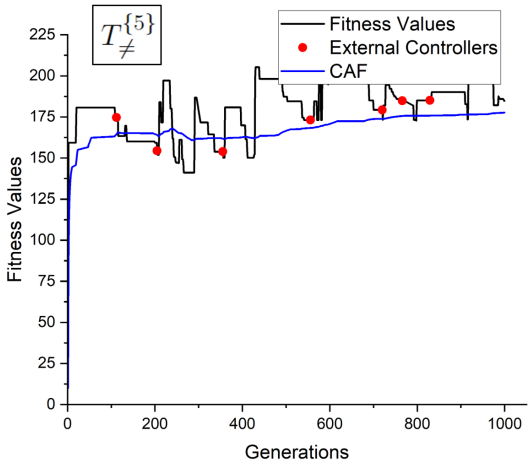


Figure 5.9: Variation in the *Controller* fitness values over generations in the case $T_{\neq}^{\{5\}}$ in E_T

comparatively lesser jitters over the generations.

Figure 5.13 shows the bar graph of the CAF values at the 1000^th generation for all the transfers performed so far for the same task. The CAF values attained by the higher order transfers (including full transfers) seem to have higher CAF values than the lower order ones. Performing higher order transfers therefore, seem to be more suited in cases when the tasks at the source and the target are the same.

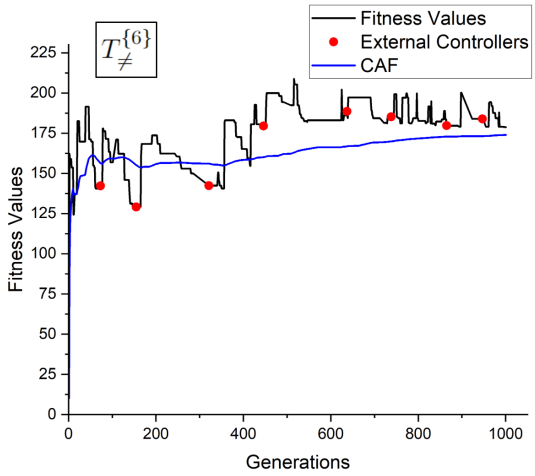


Figure 5.10: Variation in the *Controller* fitness values over generations in the case $T_{\neq}^{\{6\}}$ in E_T

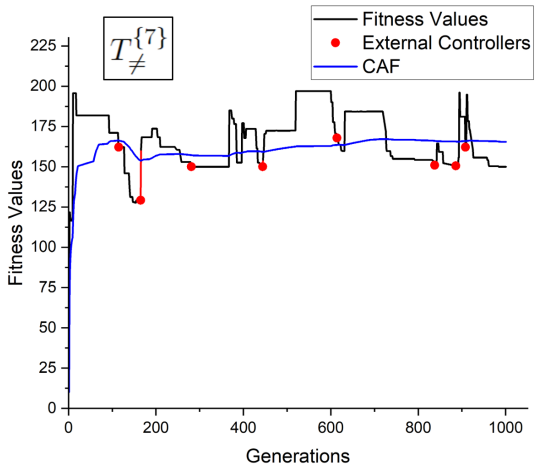


Figure 5.11: Variation in the *Controller* fitness values over generations in the case $T_{\neq}^{\{7\}}$ in E_T

5.3.2 Evolving Dissimilar Tasks $T_{\neq}$

The graphs in the figures 5.8-5.12 depict the variation of fitness values and CAF over generations after the transfer of the repertoire from E_S to E_T. Both the environments are dissimilar and evolve the dissimilar tasks of obstacle avoidance and obstacle avoidance cum phototaxis. In the current case, the controllers within the repertoire in E_T not only need to evolve and learn to perform the new task but also have to ensure their adaptation to the dissimilar environment. This makes the

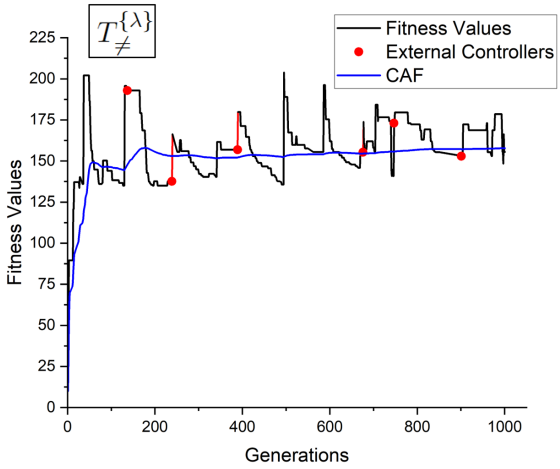


Figure 5.12: Variation in the *Controller* fitness values over generations in the case $T_{\neq}^{\{\lambda\}}$ in E_T

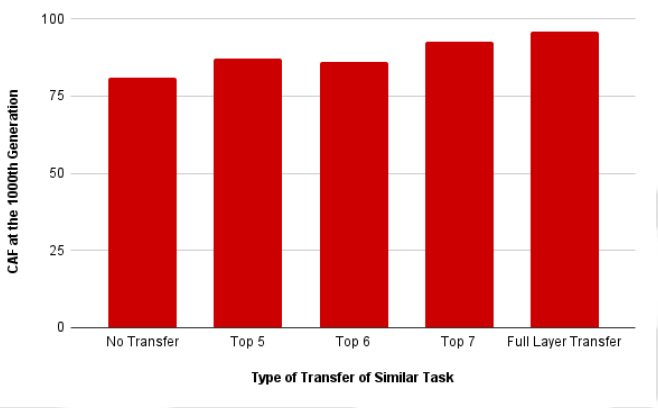


Figure 5.13: Cumulative Average Fitness at 1000^{th} generation for Similar tasks

$T_{\neq}$ scenario a more challenging one. The best controllers evolved from the shared repertoire (External repertoire Re_{Ex}) at E_T are highlighted in all graphs in figures 5.9-5.12 using tiny red circles. In the graph for $T_{\neq}^{\{0\}}$, shown in figure 5.8, since the repertoire is not transferred to the target, the controllers need to be evolved right from scratch. The challenging nature of the scenario, mentioned earlier, adds to the toil. The fitness curve is thus, not smooth and the CAF does not attain a high value.

In the cases $T_{\neq}^{\{5\}}$ and $T_{\neq}^{\{6\}}$ (figures 5.9 and 5.10), where the top 5 and 6 neurons, respectively, are transferred, a marked improvement can be observed in the fitness curves. Beyond 500 generations, both curves seem to hold on to values above 175.

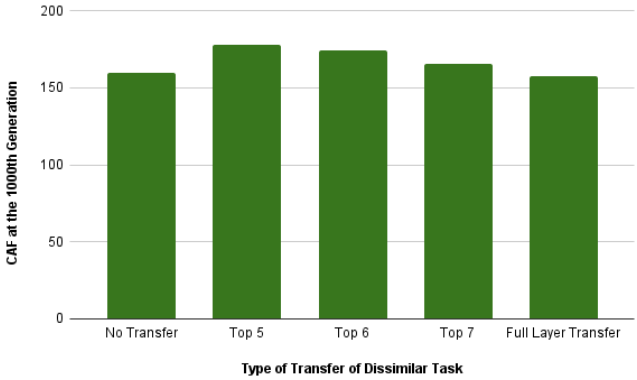


Figure 5.14: Cumulative Average Fitness at 1000^{th} generation for Dissimilar tasks

It can be noted that the $T_{\neq}^{\{6\}}$ transfer fares better than $T_{\neq}^{\{5\}}$. When the order of transfer is increased as in $T_{\neq}^{\{7\}}$ and $T_{\neq}^{\{\lambda\}}$, the fitness curve exhibits more rises and falls as compared to its lower order counterparts. After 500 generations, the fitness values seem to saturate to values less than 175. Thus, when the tasks are dissimilar, it can be clearly observed that lower order transfers (like the top 5 or 6) perform better than the higher order ones. It may be noted that the fitness values in the cases of $T_{=}$ and $T_{\neq}$ are different since corresponding equations 2.12 governing them are different.

When the E_S and E_T are evolving the dissimilar tasks, some of the neurons evolved in the E_S are not suitable to be used in the controllers within the E_T. In higher order transfers the neurons have already adapted to the task in E_S. These need to be first reset and then made to relearn the new task in E_T. This makes the fitness curves of $T_{\neq}^{\{7\}}$ and $T_{\neq}^{\{\lambda\}}$ to fluctuate more often resulting in comparatively lower fitness values than their lower order counterparts.

Figure 5.14 depicts the bar graph of the CAF values at the 1000^th generation for all the transfers performed so far for the dissimilar tasks. Contrary to the observations in the CAF values for $T_{=}$ (figure 5.13), the ones in $T_{\neq}$, attain higher CAF values for lower orders as compared to the higher order ones (including full transfers).

Performing lower order transfer in $T_{\neq}$ scenarios therefore, seems to be more beneficial when the tasks at the source and the target environments are dissimilar.

Whereas, in the cases of lower order transfers, few of the neurons that are

tuned the source task have to be adapted, and the since other neurons are random initialized, they will have to learn the task without any resetting. Since, there is no extra energy expended to reset the learning, the lower order transfer fare very well than higher order ones and also maintain a higher fitness values as observed in the graphs in figures 5.9 and 5.10. This can also be clearly understood by observing the value of CAF at 1000^th generation as depicted in figure 5.14. The CAF values attained by the lower order transfers of top 5 and top 6 ($T_{\neq}^{\{5\}}$ and $T_{\neq}^{\{6\}}$) are higher than the higher orders of transfer and also the case when there is no transfer ($T_{\neq}^{\{0\}}$).

5.4 Summary of the Chapter

This chapter captures the essence of all the work reported in the earlier chapters of this thesis and presents the same embedded in a robotic application scenario. The first and second chapters dealt with how the embodied learning can be achieved in the robots addressing the challenges elaborated in Chapter 1. These chapters also focused on how such a learning can be enhanced in the domain of neuroevolution. The third chapter focused on the concept of transfer learning in closed world problems, addressing the problem of such transfers amongst dissimilar domains. Finally, this chapter focused on applying transfer learning on embodied robots. The application described herein adapted the *iAES-HoF* algorithm proposed in Chapter 2 for evolving robot controllers in one environment and then transferring the learned information via neurons, using a modified version of the neuronal transfer learning algorithm presented in Chapter 4. In brief, the chapter provides a proof of concept on the working of the algorithms reported in earlier chapters in the realm of Immuno-inspired Embodied Life-long learning.

From the results of this chapter, it can be concluded that the transfer of *Hot* neurons of all the controllers of the source repertoire Re_S from source environment E_S to the target repertoire Re_T in the target environment E_T is beneficial in both the cases of same and dissimilar tasks. In the case, where both E_S and E_T are evolving the same tasks, higher order transfers are beneficial and accelerate the learning. Whereas, in the case of dissimilar tasks, lower order transfers are beneficial. Robotic simulations performed using *Webots* indicate that the proposed transfer learning is

beneficial even in cases where the robots in the source and target environments carry out dissimilar or new tasks. The simulations performed can be directly adapted to the real world robots, that inhabit the real environments.

6

Conclusions and Future Research Directions

6.1 Conclusions of the Thesis

The thesis aimed at achieving embodied lifelong learning in robots by addressing all the challenges elaborated in Chapter 1, in the best possible way given both the time and resources. The first contribution elaborated in Chapter 2, focuses on achieving embodied lifelong learning in a robot. The search space was divided into various active regions on the fly, governed by the parameter ϵ. A Hall of Fame of controllers evolved and maintained for every active region. An artificial neural network constituted a controller, which was evolved by neuroevolution. The parameters *Concentration* and *Resource* governed the re-selection and eviction of the controllers from an HoF. The Concentration and Resource values of the controllers were dynamically tuned based on the performance of the respective controllers and were not set *a priori*. Choosing the value of ϵ also played an important part in evolving the controllers. A high value of ϵ led to evolving large active regions and posed a challenge to evolve controllers catering to diverse antigens encountered in each of such large active regions. Whereas a lower value of ϵ lead to small active regions and switching of controllers from one active region to another without achieving the desired learning. The experiments done on both simulated and real robots established

the efficacy of the proposed Immuno-Inspired evolutionary algorithm.

In embodied lifelong learning, the robot controllers are evolved by neuroevolutionary techniques, and the Immuno-Inspired algorithm maintains the halls of fame of robot controllers. The second contribution focuses on how the neuroevolution to evolve the robot controllers can be enhanced. With every weight of the ANN, a *Mutational Puissance* was associated, and *Mutational Puissances* are increased or decreased based on the performance of the associated ANN-based controller. Instead of mutating all the weights of the ANN, the mutation of weights is guided by these Puissances. Experiments showed that a combination of random and Puissance-guided mutation leads to better learning of robot controllers.

In the following two contributions, the thesis focuses on transferring the learned information to a target environment. Before implementing the transfer in the open world of robotics, the third contribution deals with transferring the learned information from source to target in a closed world problem. The proposed Immuno-Inspired Methodology identifies the *Hot* neurons from the source ANN trained on a source dataset to the target ANN to be trained on the target dataset. The proposed neuronal transfer learning method works even when the source and the target datasets are dissimilar. The experimental results establish the efficacy of the method on both shallow and deep neural networks.

The final contribution discussed in Chapter 5, emphasizes how the work described in the earlier chapters can be integrated and applied to the realm of robotics. The work reported in this chapter is an attempt at displaying the practicality of the overall work. It combines the use of HoFs and transfer learning to show how neuroevolution can be embodied in robots so as to achieve life-long learning amongst robots. The methodology and application presented in this chapter focuses on the neuronal transfer of ANN-based robot controllers from the source HoF to the target HoF. *Hot* neurons are identified from within the controllers of all the HoF and then transferred to the target end to eventually accelerate the learning process. The experimental results indicate improved learning in the target robot controllers after the transfer is affected, even if the source and target environments of the respective robots are different and when they need to learn new tasks.

6.2 Future Research Directions and Applications

This thesis addresses the challenges of embodied lifelong learning in an online and on-board manner in robots and also provides scope for further research possibilities, some of which are elucidated below. In the first contribution, the hyperparameter ϵ was set to a certain value to divide the shape space into various active regions. Since ϵ governs the number of active regions and hence the number of HoFs, ascertaining and assigning a value to it requires to be performed empirically. As already mentioned, higher values of ϵ would mean lesser number of controllers thereby increasing the learning load on these controllers. Lower values could cause an explosion in the number of active regions and associated controllers. The associated algorithm, needs to be augmented so that ϵ is tuned in an adaptive manner based on the environment of the robot. Its value needs to be determined and also changed based on the interactions of the robot(s) with the environment and the performance of the respective controllers during run time. The algorithm could start off with a random value of ϵ and either increase or decrease it based on hypermutation [115] controlled by the performance of the controllers. In addition, it may be noted that the proposed algorithm generates active regions, all of which which, have the same size. In brief, it divides the entire search space into active regions of the same size. In the real world, such a uniform division of the search space may not always be the best option. Contiguous portions of the search space where fairly homogeneous antibodies (controllers) can cater to the problems within, could be accommodated in a large single active region (high ϵ). On the contrary, those portions which require diverse antibodies could be split into numerous small active regions (low ϵ). In other words, there needs to be a mechanism to split the entire search space into active regions of different sizes that cater to the actual requirements of these regions. In the second contribution, the Puissance-based strategy the extent of mutation of weights could be made to be adaptive in nature so that not all weights need to be mutated by the same extent. This strategy could also be extended to be used in Multi-Robot Scenarios and for problems involving multi-objective optimization.

In the last two contributions involving transfer learning, the number of top hot neurons is a hyper parameter that needs to be fixed *a priori*. Methodologies

to find the apt value of this parameter also needs to be evolved. The similarity and dissimilarity of the source and target datasets/tasks also need to be quantified to gain insights into determining the extent of such transfer. In the open world of robotics, a mechanism for the robot to *feel* a change in its environment, still remains an open challenge.

The contributions made in this thesis can be applied to scenarios where the environment changes dynamically, and embedded learning mechanisms need to keep pace with these changes while also retaining the knowledge already acquired in the past. Such locales include Automated Autonomous vehicles, which need to learn new tasks on-the-fly or dynamically, without forgetting the already learned ones, could also use the algorithms proposed in this thesis. Autonomous Ground Vehicles (AGV) in warehouses that need to *learn* to perform tasks, drones carrying out their assigned tasks, also could benefit accordingly. AGVs learning to perform tasks in the warehouses, may require to adapt to the changing environmental conditions within caused by the goods being moved in and out frequently. By incorporating the embodied lifelong learning presented in Chapter 2, AGVs can not only learn but also preserve the information learned over time. The retention and elimination of the learned behaviors could be autonomously decided by the AGVs without any external intervention. The learned behaviors from the AGVs in one warehouse can be also be transferred to other AGVs inhabiting other warehouses by adapting the works presented in Chapters 4 and 5. Since it may be always the case that the tasks at these warehouses are similar, learning of the dissimilar ones can be hastened using the proposed transfer learning techniques.

References

[1] S. Acharya, A. K. Pant, and P. K. Gyawali. Deep learning based large scale handwritten devanagari character recognition. In *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 1–6, 2015.

[2] G. L. Ada and S. G. Nossal. The clonal-selection theory. *Scientific American*, 257(2):62–69, 1987.

[3] A. Agogino, K. Stanley, and R. Miikkulainen. Online interactive neuro-evolution. *Neural Processing Letters*, 11(1):29–38, 2000.

[4] U. Aickelin and S. Cayzer. The danger theory and its application to artificial immune systems. *arXiv preprint arXiv:0801.3549*, 2008.

[5] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang. A survey on evolutionary machine learning. *Journal of the Royal Society of New Zealand*, 49(2):205–228, 2019.

[6] A. M. Andrew. Introduction to evolutionary computing. *Kybernetes*, 2004.

[7] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Inc., USA, 1996.

[8] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Inc., USA, 1996.

[9] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[10] J. Balthrop, F. Esponda, S. Forrest, M. Glickman, et al. Coverage and generalization in an artificial immune system. In *GECCO*, volume 2, pages 3–10. Citeseer, 2002.

[11] R. D. Beer and J. C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122, 1992.

[12] H. Bersini and F. J. Varela. Hints for adaptive problem solving gleaned from immune networks. In *International Conference on Parallel Problem Solving from Nature*, pages 343–354. Springer, 1990.

[13] S. Binitha, S. S. Sathya, et al. A survey of bio inspired optimization algorithms. *International journal of soft computing and engineering*, 2(2):137–151, 2012.

[14] N. Bredeche, E. Haasdijk, and A. E. Eiben. On-line, on-board evolution of robot controllers. In P. Collet, N. Monmarché, P. Legrand, M. Schoenauer, and E. Lutton, editors, *Artifical Evolution*, pages 110–121, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[15] E. S. Brunette, R. C. Flemmer, and C. L. Flemmer. A review of artificial intelligence. In *2009 4th International Conference on Autonomous Robots and Agents*, pages 385–392. Ieee, 2009.

[16] A. Camero, J. Toutouh, D. H. Stolfi, and E. Alba. Evolutionary deep learning for car park occupancy prediction in smart cities. In R. Battiti, M. Brunato, I. Kotsireas, and P. M. Pardalos, editors, *Learning and Intelligent Optimization*, pages 386–401, Cham, 2019. Springer International Publishing.

[17] A. Cangelosi, A. Greco, and S. Harnad. From robotic toil to symbolic theft: grounding transfer from entry-level to higher-level categories1. *Connection science*, 12(2):143–162, 2000.

[18] L. Chen, H.-L. Liu, K. C. Tan, and K. Li. Transfer learning-based parallel evolutionary algorithm framework for bilevel optimization. *IEEE Transactions on Evolutionary Computation*, 26(1):115–129, 2022.

[19] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

[20] A. Cully and J.-B. Mouret. Behavioral repertoire learning in robotics. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 175–182, New York, NY, USA, 2013. ACM.

[21] B. Da, A. Gupta, and Y.-S. Ong. Curbing negative influences online for seamless transfer evolutionary optimization. *IEEE Transactions on Cybernetics*, 49(12):4365–4378, 2019.

[22] B. Dahal and J. Zhan. Effective mutation and recombination for evolving convolutional networks. In *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, pages 1–6, 2020.

[23] A. Darwish. Bio-inspired computing: Algorithms review, deep analysis, and the scope of applications. *Future Computing and Informatics Journal*, 3(2):231–246, 2018.

[24] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2010.

[25] D. Dasgupta. *An Overview of Artificial Immune Systems and Their Applications*, pages 3–21. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[26] D. Dasgupta. *Artificial immune systems and their applications*. Springer Science & Business Media, 2012.

[27] D. Dasgupta and N. Attoh-Okine. Immunity-based systems: a survey. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 1, pages 369–374 vol.1, Oct 1997.

[28] D. Dasgupta, Z. Ji, and F. Gonzalez. Artificial immune system (ais) research in the last five years. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 1, pages 123–130 Vol.1, Dec 2003.

[29] D. Dasgupta, S. Yu, and F. Nino. Recent advances in artificial immune systems: models and applications. *Applied Soft Computing*, 11(2):1574–1587, 2011.

[30] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. E. G. Eiben. Evolutionary robotics: What, why, and where to. *Frontiers in Robotics and AI*, 2, 2015.

[31] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[32] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244, 2022.

[33] A. E. Eiben, J. E. Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[34] B. Evans, H. Al-Sahaf, B. Xue, and M. Zhang. Evolutionary deep learning: A genetic programming approach to image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–6, 2018.

[35] J. Farmer, N. H. Packard, and A. S. Perelson. The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 22(1):187 – 204, 1986. Proceedings of the Fifth Annual International Conference.

[36] J. D. Farmer, N. H. Packard, and A. S. Perelson. The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 22(1-3):187–204, 1986.

[37] S. G. Ficici, R. A. Watson, and J. B. Pollack. Embodied evolution: A response to challenges in evolutionary robotics. In *Proceedings of the eighth European workshop on learning robots*, pages 14–22. Citeseer, 1999.

[38] I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D. Fister. A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv:1307.4186*, 2013.

[39] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1):47–62, 2008.

[40] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 421–430. The MIT Press, 1994.

[41] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot. *Proceedings of the third international conference on Simulation of adaptive behavior: From Animals to Animats 3*, pages 421–430, 1994. D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson (eds.).

[42] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE transactions on neural networks*, 5(1):3–14, 1994.

[43] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 120–128, May 1996.

[44] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, SP '94, pages 202–, Washington, DC, USA, 1994. IEEE Computer Society.

[45] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[46] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[47] S. Friess, P. Tiňo, S. Menzel, B. Sendhoff, and X. Yao. Improving sampling in evolution strategies through mixture-based distributions built from past problem instances. In T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann, editors, *Parallel Problem Solving from Nature – PPSN XVI*, pages 583–596, Cham, 2020. Springer International Publishing.

[48] M. F. A. Gadi, X. Wang, and A. P. do Lago. Credit card fraud detection with artificial immune system. In P. J. Bentley, D. Lee, and S. Jung, editors, *Artificial Immune Systems*, pages 119–131, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[49] W. W. Godfrey, S. S. Jha, and S. B. Nair. On stigmergically controlling a population of heterogeneous mobile agents using cloning resource. In *Transactions on Computational Collective Intelligence XIV*, pages 49–70. Springer, 2014.

[50] W. W. Godfrey, S. B. Nair, and Kim Dong Hwa. Towards a dynamic emotional model. In *2009 IEEE International Symposium on Industrial Electronics*, pages 1932–1936, July 2009.

[51] D. E. Goldberg. *Genetic algorithms*. Pearson Education India, 2006.

[52] S. Golzari, S. Doraisamy, M. N. Sulaiman, N. I. Udzir, and N. M. Norowi. Artificial immune recognition system with nonlinear resource allocation method and application to traditional malay music genre classification. In *Proceedings of the 7th International Conference on Artificial Immune Systems*, ICARIS '08, pages 132–141. Springer-Verlag, 2008.

[53] M. Gong, L. Jiao, H. Du, and L. Bo. Multiobjective immune algorithm with nondominated neighbor-based selection. *Evolutionary Computation*, 16(2):225–255, June 2008.

[54] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris. Spottune: Transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[55] E. Hart. Towards lifelong learning in optimisation algorithms. In *Proceedings of the 9th International Joint Conference on Computational Intelligence*, volume 1, pages 7–9, 2017.

[56] E. Hart and J. Timmis. Application areas of ais: The past, the present and the future. *Applied soft computing*, 8(1):191–201, 2008.

[57] J. H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[58] Y. Hu and W. Luo. Robotic immune systems: An architecture. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1098–1103, 2018.

[59] J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[60] A. Ishiguro, R. Watanabe, and Y. Uchikawa. An immunological approach to dynamic behavior control for autonomous mobile robots. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 1, pages 495–500 vol.1, Aug 1995.

[61] N. K. Jerne. Towards a network theory of the immune system. *Ann. Immunol.*, 125:373–389, 1974.

[62] N. K. Jerne and J. Cocteau. Idiotypic networks and other preconceived ideas. *Immunological reviews*, 79(1):5–24, 1984.

[63] S. S. Jha, W. W. Godfrey, and S. B. Nair. Stigmergy-based synchronization of a sequence of tasks in a network of asynchronous nodes. *Cybernetics and Systems*, 45(5):373–406, 2014.

[64] S. S. Jha, K. Shrivastava, and S. B. Nair. On emulating real-world distributed intelligence using mobile agent based localized idiotypic networks. In R. Prasath and T. Kathirvalavakumar, editors, *Mining Intelligence and Knowledge Exploration*, pages 487–498, Cham, 2013. Springer International Publishing.

[65] S. S. Jha, K. Shrivastava, and S. B. Nair. On emulating real-world distributed intelligence using mobile agent based localized idiotypic networks. In R. Prasath and T. Kathirvalavakumar, editors, *Mining Intelligence and Knowledge Exploration*, pages 487–498, Cham, 2013. Springer International Publishing.

[66] J. Kelsey and J. Timmis. Immune inspired somatic contiguous hypermutation for function optimisation. In E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation — GECCO 2003*, pages 207–218, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[67] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[68] J. O. Kephart et al. A biologically inspired immune system for computers. In *Artificial Life IV: proceedings of the fourth international workshop on the synthesis and simulation of living systems*, pages 130–139, 1994.

[69] T. B. Kepler and A. S. Perelson. Somatic hypermutation in b cells: an optimal control treatment. *Journal of theoretical biology*, 164(1):37–64, 1993.

[70] E. Klarreich et al. Inspired by immunity. *Nature*, 415(6871):468–470, 2002.

[71] S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.

[72] D. D. Kulkarni and S. B. Nair. *Mutational Puissance Assisted Neuroevolution*, page 1841–1848. Association for Computing Machinery, New York, NY, USA, 2020.

[73] D. D. Kulkarni and S. B. Nair. An immuno-inspired transfer learning paradigm. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 2515–2522, 2021.

[74] D. D. Kulkarni, T. Semwal, and S. B. Nair. Immuno-inspired management of halls of fame for embodied evolution. *Swarm and Evolutionary Computation*, 70:101054, 2022.

[75] W. B. Langdon and R. Poli. *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[76] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[77] D.-W. Lee and K.-B. Sim. Artificial immune network-based cooperative control in collective autonomous mobile robots. In *Proceedings 6th IEEE International Workshop on Robot and Human Communication. RO-MAN'97 SENDAI*, pages 58–63, Sept 1997.

[78] J. Lehman, J. Chen, J. Clune, and K. O. Stanley. Safe mutations for deep and recurrent neural networks through output gradients. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 117–124, 2018.

[79] H. Li, C. Wang, and A. Wang. Medical image registration based on more features and artificial immune algorithm. In *Artificial Intelligence, 2009. JCAI'09. International Joint Conference on*, pages 469–472. IEEE, 2009.

[80] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 97–105. JMLR.org, 2015.

[81] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang. Transfer learning using computational intelligence: A survey. *Knowledge-Based Systems*, 80:14–23, 2015. 25th anniversary of Knowledge-Based Systems.

[82] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang. Transfer learning using computational intelligence: A survey. *Knowledge-Based Systems*, 80:14–23, 2015. 25th anniversary of Knowledge-Based Systems.

[83] X. Ma, Q. Chen, Y. Yu, Y. Sun, L. Ma, and Z. Zhu. A two-level transfer learning algorithm for evolutionary multitasking. *Frontiers in Neuroscience*, 13:1408, 2020.

[84] P. Matzinger. The danger model: a renewed sense of self. *science*, 296(5566):301–305, 2002.

[85] P. E. McKnight and J. Najab. Mann-whitney u test. *The Corsini encyclopedia of psychology*, pages 1–1, 2010.

[86] Z. Michalewicz. *Evolution Strategies and Other Methods*, pages 159–177. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[87] O. Michel. Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004.

[88] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[89] M. B. Montaner and A. Ramirez-Serrano. Fuzzy knowledge-based controller design for autonomous robot navigation. *Expert Systems with Applications*, 14(1-2):179–186, 1998.

[90] J.-B. Mouret and G. Maguire. Quality diversity for multi-task optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 121–129, New York, NY, USA, 2020. Association for Computing Machinery.

[91] S. B. Nair, W. W. Godfrey, and D. H. Kim. On realizing a multi-agent emotion engine. *International Journal of Synthetic Emotions (IJSE)*, 2(2):1–27, 2011.

[92] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345 – 370, 2009.

[93] G. Nicolai and R. J. Hilderman. No-limit texas hold'em poker agents created with evolutionary neural networks. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 125–131, Sep. 2009.

[94] S. Nikhil, T. Semwal, and S. B. Nair. Immuno-inspired behaviour adaptation in multi-robot systems. In *2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016, Budapest, Hungary, October 9-12, 2016*, pages 3199–3204, 2016.

[95] M. Nogueira, C. Cotta, and A. J. Fernández-Leiva. An analysis of hall-of-fame strategies in competitive coevolutionary algorithms for self-learning in rts games. In G. Nicosia and P. Pardalos, editors, *Learning and Intelligent Optimization*, pages 174–188, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[96] S. Nolfi. Evolutionary robotics: Exploiting the full power of self-organization. 10, 08 1998.

[97] S. Nolfi, J. Bongard, P. Husbands, and D. Floreano. *Evolutionary Robotics*, pages 2035–2068. Springer International Publishing, Cham, 2016.

[98] S. Nolfi and D. Floreano. Synthesis of autonomous robots through evolution. *Trends in Cognitive Sciences*, 6(1):31–37, 2002.

[99] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[100] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[101] A. Pentina and C. H. Lampert. Lifelong learning with non-iid tasks. *Advances in Neural Information Processing Systems*, 28, 2015.

[102] R. Pfeifer, M. Lungarella, and F. Iida. Self-organization, embodiment, and biologically inspired robotics. *Science*, 318(5853):1088–1093, 2007.

[103] J. K. Pugh, L. B. Soros, and K. O. Stanley. An extended study of quality diversity algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, page 19–20, New York, NY, USA, 2016. Association for Computing Machinery.

[104] A. Raza and B. R. Fernandez. Immuno-inspired heterogeneous mobile robotic systems. In *49th IEEE Conference on Decision and Control (CDC)*, pages 7178–7183. IEEE, 2010.

[105] A. Raza and B. R. Fernandez. Immuno-inspired robotic applications: a review. *Applied Soft Computing*, 37:490–505, 2015.

[106] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, volume 898, pages 1–4, 2005.

[107] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[108] N. Roy, I. Posner, T. Barfoot, P. Beaudoin, Y. Bengio, J. Bohg, O. Brock, I. Depatie, D. Fox, D. Koditschek, et al. From machine learning to robotics: Challenges and opportunities for embodied intelligence. *arXiv preprint arXiv:2110.15245*, 2021.

[109] Rui Yu and Zong-yuan Mao. A new idiotypic immune network based pid controller design. In *2010 8th World Congress on Intelligent Control and Automation*, pages 2486–2491, 2010.

[110] T. Semwal, D. D. Kulkarni, and S. B. Nair. On an immuno-inspired distributed, embodied action-evolution cum selection algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 141–148, New York, NY, USA, 2018. ACM.

[111] T. Shimooka and K. Shimizu. Idiotypic network model for feature extraction in pattern recognition – effect of diffusion of antibody. In V. Palade, R. J. Howlett, and L. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, pages 511–518, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[112] F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen. Open issues in evolutionary robotics. *Evolutionary Computation*, 24(2):205–236, 2016. PMID: 26581015.

[113] F. Silva, P. Urbano, and A. L. Christensen. Adaptation of robot behaviour through online evolution and neuromodulated learning. In J. Pavón, N. D. Duque-Méndez, and R. Fuentes-Fernández, editors, *Advances in Artificial Intelligence – IBERAMIA 2012*, pages 300–309, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[114] C. T. Singh and S. B. Nair. An artificial immune system for a multiagent robotics system. In *Proc. of the 4th World Enformatika International Conference on Automation Robotics and Autonomous Systems (ARAS 2005)*, pages 308–311, 2005.

[115] C. T. Singh and S. B. Nair. An artificial immune system for a multiagent robotics system. In *Proc. of the 4th World Enformatika International Conference on Automation Robotics and Autonomous Systems (ARAS 2005)*, pages 308–311, 2005.

[116] W. M. Spears and V. Anand. A study of crossover operators in genetic programming. In *International symposium on methodologies for intelligent systems*, pages 409–418. Springer, 1991.

[117] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312, 2016.

[118] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. In V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 270–279, Cham, 2018. Springer International Publishing.

[119] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[120] J. Timmis, M. Neal, and J. Hunt. An artificial immune system for data analysis. *Biosystems*, 55(1-3):143–150, 2000.

[121] I. N. Vieira, B. S. L. P. de Lima, and B. P. Jacob. Optimization of steel catenary risers for offshore oil production using artificial immune system. In P. J. Bentley, D. Lee, and S. Jung, editors, *Artificial Immune Systems*, pages 254–265, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[122] B. Wang, J. Mendez, M. Cai, and E. Eaton. Transfer learning via minimizing the performance gap between domains. *Advances in Neural Information Processing Systems*, 32, 2019.

[123] Z. Wang, Z. Dai, B. Poczos, and J. Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[124] Y. Watanabe, A. Ishiguro, and Y. Uchikawa. *Decentralized Behavior Arbitration Mechanism for Autonomous Mobile Robot Using Immune Network*, pages 187–209. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[125] A. Watkins and J. Timmis. Artificial immune recognition system (airs): Revisions and refinements. In *AISB 2004 Convention*, page 18, 2004.

[126] Y. WEI, Y. Zhang, J. Huang, and Q. Yang. Transfer learning via learning to transfer. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5085–5094, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[127] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[128] A. Whitbrook, U. Aickelin, and J. Garibaldi. An idiotypic immune network as a short-term learning architecture for mobile robots. In P. J. Bentley, D. Lee, and S. Jung, editors, *Artificial Immune Systems*, pages 266–278, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[129] A. M. Whitbrook, U. Aickelin, and J. M. Garibaldi. Idiotypic immune networks in mobile-robot control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(6):1581–1598, Dec 2007.

[130] S. Whitelam, V. Selin, S.-W. Park, and I. Tamblyn. Correspondence between neuroevolution and gradient descent. *Nature communications*, 12(1):1–10, 2021.

[131] X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, and M. Karamanoglu. *Swarm intelligence and bio-inspired computation: theory and applications*. Newnes, 2013.

[132] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, Sept 1999.

[133] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[134] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021.

Publications

Relevant to thesis

Conferences

1. **Divya D. Kulkarni** and Shivashankar B. Nair. "Mutational puissance assisted neuroevolution." In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pp. 1841-1848. 2020

2. **Divya D. Kulkarni** and Shivashankar B. Nair. "An Immuno-Inspired Transfer Learning Paradigm." In 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 2515-2522. IEEE, 2021.

3. **Divya D. Kulkarni** and Shivashankar B. Nair. "Mutational puissance assisted neuroevolution." In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (Accepted at GECCO 2023, Lisbon, Portugal)

Journals

4. **Divya D. Kulkarni**, Tushar Semwal, and Shivashankar B. Nair. "Immuno-inspired management of halls of fame for embodied evolution." Swarm and Evolutionary Computation 70 (2022): 101054. Elsevier.

Others

5. Tushar Semwal, **Divya D. Kulkarni** and Shivashankar B. Nair. "On an immuno-inspired distributed, embodied action-evolution cum selection algorithm." In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 141-148. 2018.

6. Ishita, **Divya D. Kulkarni**, Tushar Semwal, and Shivashankar B. Nair. "On securing mobile agents using blockchain technology." In 2019 Second

International Conference on Advanced Computational and Communication Paradigms (ICACCP), pp. 1-5. IEEE, 2019.

7. **Divya D. Kulkarni** and Shivashankar B. Nair. "Agrilogistics-A Genetic Programming Based Approach." In International Conference on Society with Future: Smart and Liveable Cities, pp. 83-96. Springer, Cham, 2019.

8. Akul Agrawal, **Divya D. Kulkarni**, and Shivashankar B. Nair. "On Decentralizing Federated Learning." In 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 1590-1595. IEEE, 2020.

9. Jayprakash S. Nair, **Divya D. Kulkarni**, Ajitem Joshi, and Sruthy Suresh , "On Decentralizing Federated Reinforcement Learning in Multi-Robot Scenarios," 2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Ioannina, Greece, 2022, pp. 1-8.

Presented and To Appear Online

10. Gayathri Rangu, **Divya D. Kulkarni**, J. S. Nair and Shivashankar B. Nair, A Hybrid Federated Reinforcement Learning Approach for Networked Robots, 1st international Conference on SCI/Tech and Engineering, Imphal, 17-18th Feb. 2023 (To appear in Springer Nature)

Appendice

.1 List of Abbreviations

Terms	Abbreviations
ANN	Artificial Neural Network
ER	Evolutionary Robotics
EA	Evolutionary Algorithm
BIS	Biological Immune System
AIS	Artificial Immune System
HoF	Hall of Fame
$iAES$	Immuno-Inspired Action-Evolution cum Selection Algorithm
IN	Immune Network
Ctr	Controller that drives the robot
ID	Identifier of an Antibody
$iAES - HoF$	Immuno-Inspired Action-Evolution cum Selection Algorithm with Hall-of-Fame
$IRSensor$	Infrared Sensor
OA	Obstacle Avoidance
$PT - OA$	Phototaxis cum-obstacle-avoidance
CAF	Cumulative Average Fitness
CNN	Convolutional Neural Networks
$AW - M$	All Weights mutated
$RMW - M$	Random Number of Weights Mutated
LIN	Local Idiotypic Network
TL	Transfer Learning
$MNIST$	Modified National Institute of Standards and Technology database
$KMNIST$	Kuzushiji-MNIST
$NeEvoT$	NeuroEvotionary-Transfer Learning Algorithm

.2 Glossary of Terms

Terms	Description
Active Region	Small region in the search space of a robot determined by ϵ
Affinity	Extent of binding between an antibody and an antigen
Antibody	Proteins secreted by living organisms that will tag antigens for removal
Antigen	A foreign entity attacking a living entity.
Artificial Immune System (AIS)	Computational Model inspired from BIS
Biological Immune System (BIS)	Immune System present in the Vertebrates
Champ	The best controller of an active region
Cross Reactivity Threshold (ϵ)	A parameter that divides the search space of the robot in to many small regions.
Epitope	Part of antigen where an antibody binds
Ex-Champ	The controller which was the *Champ* before, and now residing in the *HoF*.
Fitness Value	The quality of the controller
Hall of Fame (HoF)	Set of controllers of every *Active Region*
Hot Neurons	Neurons that have significantly contributed to the learning
Mutant	The controller that tries to outperform the *Champ*.
Mutational Puissance	Associated with the weights of an ANN and guiding the learning process
Paratope	Binding site of antibody, that binds with epitope of matching antigen
Temperature of the Neurons	Extent of the hotness of the neurons