Rochester Institute of Technology

RIT Scholar Works

Theses

3-2019

Relational Neurogenesis for Lifelong Learning Agents

Tej Pandit
tnp5438@rit.edu

Follow this and additional works at: <https://scholarworks.rit.edu/theses>

Recommended Citation

Pandit, Tej, "Relational Neurogenesis for Lifelong Learning Agents" (2019). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.

Relational Neurogenesis for Lifelong Learning Agents

TEJ PANDIT

Relational Neurogenesis for Lifelong Learning Agents

TEJ PANDIT
March 2019

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | KATE GLEASON
College of ENGINEERING

Department of Computer Engineering

Relational Neurogenesis for Lifelong Learning Agents

TEJ PANDIT

Committee Approval:

Dr. Dhireesha Kudithipudi *Advisor*
Professor

Date

Dr. Raymond Ptucha
Assistant Professor

Date

Dr. Cory Merkel
Assistant Professor

Date

Acknowledgments

This thesis is made possible through continued reinforcement from my amazing friends in the Nu.AI lab and my guide Dr. Dhireesha Kudithipudi who have collectively been my family and support system here at RIT.

The knowledge I have gathered in order to compile this work can be attributed to all my teachers who have created a wonderful and conducive learning environment (for lifelong learning) and all the hard work of researchers who have brought the field of artificial intelligence to where it is today.

The unseen efforts which bring me to this moment are those of my parents, without whom, none of this would have been possible anyways.

To Science
And both my grandfathers
who made learning and education fun

Abstract

Reinforcement learning systems have shown tremendous potential in being able to model meritorious behavior in virtual agents and robots. The ability to learn through continuous reinforcement and interaction with an environment negates the requirement of painstakingly curated datasets and hand crafted features. However, the ability to learn multiple tasks in a sequential manner, referred to as lifelong or continual learning, remains unresolved.

The search for lifelong learning algorithms creates the foundation for this work. While there has been much research conducted in supervised learning domains under lifelong learning, the reinforced lifelong learning domain remains open for much exploration. Furthermore, current implementations either concentrate on preserving information in fixed capacity networks, or propose incrementally growing networks which randomly search through an unconstrained solution space.

In order to develop a comprehensive lifelong learning algorithm, it seems essential to amalgamate these approaches into a condensed algorithm which can perform both neuroevolution and constrict network growth automatically.

This thesis proposes a novel algorithm for continual learning using neurogenesis in reinforcement learning agents. It builds upon existing neuroevolutionary techniques, and incorporates several new mechanisms for limiting the memory resources while expanding neural network learning capacity. The algorithm is tested on a custom set of sequential virtual environments which emulate several meaningful scenarios for intellectually down-scaled species.

Additionally, a library for connecting an unconstrained range of machine learning tools, in a variety of programming languages to the Unity3D simulation engine for the development of future learning algorithms and environments, is also proposed.

Contents

Signature Sheet	i
Acknowledgments	ii
Dedication	iii
Abstract	iv
Table of Contents	v
List of Figures	vii
List of Tables	x
Acronyms	xi
1 Introduction	2
1.1 Motivation	2
1.2 Preview and Contributions	4
1.3 Document Structure	8
2 Background	10
2.1 Previous Research	10
2.1.1 Q-Learning	11
2.1.2 Deep Q-Learning	13
2.1.3 NeuroEvolution	15
2.1.4 Continual Reinforcement Learning	17
2.1.5 Addressing Catastrophic Forgetting	18
2.2 DARPA L2M Initiative	19
3 Related Work	21
3.1 Reinforcement Learning	21
3.2 Lifelong Learning	22
3.3 Neurogenesis, Synaptogenesis and Neuroevolution	23

4	Algorithmic Specification	25
4.1	NEAT: Neuroevolution of Augmenting Topologies	25
4.2	Relational Neurogenesis	28
4.2.1	Mechanisms for Network Growth	29
4.2.2	Mechanism for Preservation of Information	48
5	Test Environments	51
5.1	Unity Engine	51
5.2	Description of Environments	52
5.2.1	Environment Set - A: Spider Survival	52
5.2.2	Environment Set - B: Forest Fire Rescue	60
5.3	Communication Protocol Library	67
6	Results	69
6.1	Independent Learning	70
6.1.1	Independent Environment Set A	71
6.1.2	Independent Environment Set B	72
6.2	Continual Learning	77
6.2.1	Continual Environment Set A	77
6.2.2	Continual Environment Set B	79
6.3	Combined Learning	81
7	Conclusion	82
7.1	Summary	82
7.2	Applications and Portability	83
7.3	Future Work	84
	Bibliography	87
	Glossary	90

List of Figures

4.1	Base Reinforcement Learning Algorithm Operation	29
4.2	Relational Neurogenesis Algorithm: PHASE-1	31
4.4	Calculation of plateau gradients for determining need for neurogenesis due to plateauing merit	34
4.5	Calculation of weight metrics for determining need for neurogenesis due to the challenging nature of the environment presenting opposing concepts to be learned.	36
4.6	Calculation of weight metrics for determining need for neurogenesis due to a low margin of difference between output confidence values.	39
5.1	Sensory Perception Zones for Auditory, Vibratory, Olfactory and Temperature based senses. The degree of sensory impact is divided into 3 zones, which scale up the sensory input by a factor of 10 per zone as you come closer to the spider.	53
5.2	Spider Visual System, which highlights the 5 LOS trajectories which the spider can see at one time.	54
5.3	Environment - A1: Food. Cockroaches serve as food, and the spider must locate the dead cockroaches in a timely manner in order to obtain a commensurate reward.	55
5.4	Environment - A2: Fire. Flames spawn at random points within the maze. The spider must identify these regions and remain close to them, however, must learn not to actually touch the flames.	56
5.5	Environment - A3: Foe. A predatory spider will try to eat the host spider. The spider must learn to stay out of its LOS by identifying the best paths in the map, any safe zones or learn to stalk the predator from behind (it's blind side).	58
5.6	Environment - A4: Friend. Another host AI agent (another spider) will spawn alongside the primary agent. Although only 1 of the 2 will be evaluated, both will operate under the same criteria and be controlled by the same algorithm. The goal being to observe the collective survival behavior of these agents without explicit directives.	59

5.7	A sample forest environment. Generated terrain and trees. Trees are distributed by clusters, most of which will be set on fire as designated as hot-zones.	61
5.8	Simulated Drone system with some sensors. This is the agent used in environment set B (Forest Fire). The major sensor systems are highlighted in the diagram.	62
5.9	RAW frames captured from the drone shows data from the FLIR camera as well as the RGB camera. The cumulative effect covers both visible and IR spectra and thus provides more robust human detection. The human body temperature range is very stable and thus easier to locate using the FLIR camera. It falls comfortably between the temperature of ground/trees and the temperature of the flames.	63
5.10	Environment - B1: Fire. The drone agent explores the forest during a forest fire with the goal to cover as much area as possible in the shortest possible time. The drone must learn to avoid overheating by balancing its time between hot and cool zones.	64
5.11	Environment - B2: Alert. The drone agent explores the forest in search of lost/trapped people. It is seen here identifying the people in front of it. It is rewarded for every person that it finds.	65
5.12	Environment - B3: Rescue. The drone is seen rescuing 2 people by back tracking and leading them out of the fire and to the nearest fire-engine.	65
5.13	Environment - B4: Team. As each drone is limited to rescuing only 2 humans at a time, this environment explores the use of 4 agents. Upon locating large groups of people, the drone signals other drones to assist by relaying the GPS location information to them.	66
6.1	Training Environment - A1 Independently.	71
6.2	Training Environment - A2 Independently.	72
6.3	Training Environment - A3 Independently.	73
6.4	Training Environment - A4 Independently.	73
6.5	Testing results on Environment Set A.	73
6.6	Training Environment - B1 Independently.	74
6.7	Training Environment - B2 Independently.	75
6.8	Training Environment - B3 Independently.	76
6.9	Training Environment - B4 Independently.	76
6.10	Testing results on Environment Set B.	77

6.11	Testing Environment Set A - Continually.	78
6.12	Testing Environment Set A - Continually. [Before and after limiting the network size of the EWC network]	78
6.13	Network Size [no of nodes] of algorithms trained on Environment Set A.	79
6.14	Testing Environment Set B - Continually.	80
6.15	Network Size [no of nodes] of algorithms trained on Environment Set B.	80
7.1	Nu.AI lab robot in real life (left) vs simulation (right).	86
7.2	Simulated environment of the Nu.AI Lab at RIT.	86

List of Tables

4.1	User-Defined Parameters for Relational Neurogenesis	50
-----	---	----

Acronyms

2D

Two Spatial Dimensions

3D

Three Spatial Dimensions

AGI

Artificial General Intelligence

ANN

Artificial Neural Network

DQN

Deep Q-Learning

DRL

Deep Reinforcement Learning

GPU

Graphics Processing Unit

ILP

Inductive Logic Programming

KL-divergence

Kullback-Leibler Divergence

LOS

Line of Sight

NEAT

Neuroevolution of Augmenting Topologies

NES

Natural Evolution Strategies

PPO

Proximal Policy Optimization

RRL

Relational Reinforcement Learning

TRPO

Trust Region Policy Optimization

TWEANN

Topology and Weight Evolving Neural Networks

Chapter 1

Introduction

1.1 Motivation

In the search for artificial general intelligence (AGI), one must study and reflect upon the processes responsible for the only available proof of existence for general intelligence, the human mind. In the context of this work, it is the human mind and the brain's architecture which is responsible for this intelligence. This is because, although the architecture does in fact support and facilitate intelligent behavior, the underlying algorithmic process behind those operations/thoughts are of equal importance to this field as is the brain's physical structure itself. It would be a premature conjecture to believe that AGI is limited to only one type of architecture, i.e. the one exhibited by the human brain. There are theories to believe that efficient lifelong learning mechanisms can manifest in diverse types of systems. This view motivates the scientific community to reach beyond the daunting shadow cast by the brain's nervous system, to obtain an alternative solution to its mysterious operation. This urges me to believe that the future of this field will not simply be limited to the mimicry of nature. That being said, the mimicry of nature is not straightforward. Although we should not limit ourselves to a purely biologically inspired model, there is no harm in adopting key aspects of the brain's functionality. Furthermore, we can adapt these mechanisms to suit our digital implementations, and thereby overcome

the fragile nature of a biological system, while maintaining its robust operability.

The process of lifelong learning, the ability to learn continually, without forgetting that which has already been learned, plays a key role in defining an AGI. This work tries to take a step towards achieving, not quite lifelong, but continual learning. It is inspired by two interesting phenomena observed in the human brain, one physical and the other algorithmic, and a third influenced by genetic evolution. Neurogenesis, the prior, is a subtle process by which the brain introduces and integrates new neurons into its existing learning framework. Reinforcement learning, the latter, is a method of learning concepts through repeated trials followed by meaningful feedback. Both processes offer significant scope for exploration and show great promise of being able to improve the performance of an agent interacting with an environment. Lastly, this agent undergoes evolution towards producing the next generation of agents, who share many properties of their predecessors, and are yet still different, unique and filled with much promise, this process is based on principles of genetics. Thus, we incorporate these mechanisms into an agent as we train it to survive multiple challenging virtual environments.

Lifelong learning agents have the potential to transform into helpful virtual assistants and robotic aids. Their interaction with human beings is a key aspect of their performance. To facilitate this interaction, the agents should exhibit more relatable behavior. This further motivates research to draw parallels between nature and technology, such that we can integrate its behavior into artificial agents and correlate it with our own understanding of intelligent behavior. State of the art reinforcement learning systems have already outperformed some of the highest human metrics in specific tasks [1]. It will be interesting to see whether lifelong learning mechanisms will be able to surpass their human counterparts someday as well.

1.2 Preview and Contributions

The field of machine learning has come a long way over the last 30 years. Considerable work carried out under the banner of artificial intelligence has led to significant advances in machine learning theory and algorithms. In the hope of achieving some form of intelligence in a non-biological system, scientists have put forth several different models, which try to explain the various processes required to form an intelligent system. The scientific community has always debated on a proper definition of the term "intelligence and have been unsuccessful in creating a universal measurement system for the same. The most common definition of intelligence, the ability to acquire and apply knowledge and skills", does not do justice to the various other aspects of intelligence such as judgment, comprehension, and reasoning to name a few.

To simplify this problem for machine learning, which is still in its teething phase, we have adopted the policy of considering the efficiency of any algorithm/system performing a task as a measure of the intelligence of that system. Although this works well for supervised learning tasks, where there is a distinct separation between the process which is learning and the process which is evaluating the performance, in a truly unsupervised environment (like the outside world which we live in and interact with every day), there is no supervisory body which knows the solution to all the problems which we face. And if there is, it does not interact directly. This causes us to form our own supervisory signals.

The foundations of a branch of machine learning, known as reinforcement learning, are based on the formation of these aforementioned supervisory signals. Reinforcement is primarily a term used in behavioral psychology to describe the phenomenon where a consequence that will strengthen an organism's future behavior is preceded by a specific stimulus. Simply put, the nature of the reward for performing a particular action will decide the likelihood of that action being repeated for the same set

of circumstances.

In reinforcement learning, an agent (machine emulating an organism) interacts with an environment (real or virtual surroundings), and based off that interaction, tries to optimize its ability to perform a task within that environment. Essentially, the agent learns to supervise its own behavior to match the behavior considered most rewarding in that environment. Although one might argue that the environment itself acts like a supervisory body, it can be noted that the environment is not aware of the solution to the task, there might not even exist a solution to the task. Thus, reinforcement learning has been adopted as an efficient training mechanism for certain complex tasks, where the optimum solution is either undetermined or dependent on an incalculable number of variables. In recent years, it has shown remarkable results in its ability to learn challenging games such as "Go" [2] and complex primordial tasks like "walking" [3].

However, a common problem faced by all areas of machine learning is the ability to learn continually [4]. This process is also termed "lifelong learning". Most algorithms are designed to accomplish a particular task, whereas in machine learning, an algorithm/model is considered more useful if it can generalize its performance across multiple tasks. However, the algorithms ability to perform multiple tasks is usually evaluated on each task independently. Lifelong learning highlights the need for an algorithm which can learn multiple tasks continually, and yet retain its ability to perform any prior learned task without retraining or memory refreshment/restoration. Ideally, a lifelong learning machine should be able to learn an infinite number of tasks, and although this is the ultimate goal, due to current limited-resource computing architectures, we are forced to limit the number of tasks. We instead focus on the ability of an algorithm to learn multiple tasks within constrained resources.

There are obvious hurdles posed by lifelong learning, network capacity and retention. As the number of non-overlapping tasks to be learned grows, the network

capacity needed to learn those new tasks increases. A simple solution is to maintain separate networks for each task, but this leads to a linear growth of total memory requirements, which although acceptable is not ideal, optimal or efficient. This also demands that there be a supervisory network which provides context for switching between networks. As many tasks share commonalities, independent networks will remodel these commonalities instead of sharing them. The other solution is to maintain a single large network and learn all the tasks on that network itself. This can lead to network saturation and eventually catastrophic forgetting, the phenomenon which occurs when learning a new task causes the network to forget a previously learned task. Both problems fundamentally stem from limitations imposed on the system by way of information theory. This dictates the maximum information which can be stored in a network of set capacity. This leads us to the obvious solution for solving this crisis, which is to increase the network capacity.

Over the last few years, several attempts have been made to provide solutions for growing network capacity. These methods encompass both supervised and unsupervised tasks. Some involve merging multiple smaller learned networks into a large network, or a single reduced small sized network, while other algorithms tailor the learning mechanism to modulate its effects while switching tasks. A well-known and yet less explored method involves adding neurons periodically to the network to increase its capacity. This biologically inspired process, as studies of the human brain presumably show similar behavior [5], is known as "Neurogenesis". This work hopes to tackle the two aforementioned hurdles faced by lifelong learning mechanisms by demonstrating an efficient implementation of neurogenesis within a reinforcement learning agent.

Much like previous work [6] [7] [1] [8] [9], the agent will be exposed to multiple virtual environments and varied tasks/challenges. However, these environments are modelled to simulate real world challenges faced by smaller organisms, and un-

like some previous work [10] [11] [12] [13], the agent will be unaware of when a task/challenge is switched out. This is where we propose a context-aware continual relational neurogenesis model. To elucidate, the proposed solution entails a dynamically growing graph node network, where each connection is explicitly modelled into a layer of nodes. This allows for fine grained control over the connections and prevents oversaturation of network parameters. The network growth is a function of the performance of the agent in the current task and the uniqueness of the input. This allows the network to be essentially aware of the task at hand, and whether it is, a new task or not, allowing for a decision to be made of whether neurogenesis should take place or not. This also addresses the issue of supervisory signals mentioned earlier on. Furthermore, the degree of uniqueness of an input would affect the rate of neurogenesis directly, thus only producing new nodes/neurons as required, thereby curbing network growth. Finally, the graph node nature of the network allows neurogenesis to add neurons at any position within the network. This is done by evaluating each activated neuron, along the activation path (from input to output) for every unique/new input and placing the new neuron only when and where a task diverges from a previous established task.

To model temporal features, this work proposes the use of special temporal activation matrices which average the direction and magnitude of a neuron’s activation over time, and use this property as a feature to track the uniqueness of inputs and thereby tasks. This functionality plays a bigger role in learning information over varied timescales ahead. Reinforcement learning mechanisms model the most optimum future action based on current inputs and the previous state, however, this puts a heavy toll on the traditional state matrix as it needs to model several relationships between states. This problem remains constrained in a discrete state space, however, real world scenarios involving continuous state spaces cause the problem to quickly grow out of hand.

1.3 Document Structure

Chapter 1 introduces the motivations driving this direction of research, and addresses the various aspects touched upon by this work.

Chapter 2 discusses the previous work conducted in the field which contributes towards this research. Here, background knowledge is delineated which is essential towards understanding the contributions of this work. The various research conducted under different specializations of lifelong learning and machine learning are discussed and key contributions are highlighted in order to show similarities or contradictions of this work and previous work. These are however noted in Chapter 4, and only the relevant key-point explanations are presented in Chapter 2.

Chapter 3 provides a brief overview about the foundational fields being investigated by the algorithm.

Chapter 4 explains the proposed algorithm for Relational Neurogenesis and other supporting algorithms integrated into the algorithm. It is a structurally technical module with the algorithmic specification and justification.

Chapter 5 covers the setup and details about the various simulation environments on which the proposed algorithm is tested. It also contains specifications about the Simulation Engine Software being used and its features. Further, this section also contained the technical specification of another novel contribution, the communication library, connecting the programming code to the simulation environment.

Chapter 6 is an indicative walkthrough of the algorithm, with step-by-step illustrations debugging through an iteration of the algorithm. It provides insight connecting the algorithms operation to the agent in the environment.

Chapter 7 provides the results obtained by the agent in the various environments. It also compares the results with state-of-the-art algorithms.

Chapter 8 discusses the nature of the results and investigates the intricacies of the

algorithm. Shows the analysis of the algorithm when used for continual learning.

Chapter 9 summarizes the work. It also discusses the scalability and portability of the algorithm. Finally, the possible future extensions of the are proposed and discussed briefly, and also several practical applications of this proposed algorithm are suggested.

Chapter 2

Background

2.1 Previous Research

Lifelong learning can be seen as an enhancement of machine learning algorithms' capabilities which allow tasks arriving in a sequence to be learned, retained and recalled irrespective of their order or nature. As there is no definitive model or hypothesis, as of this writing, about how biological organisms actually tackle this process, the last few decades have been filled with attempts to demonstrate artificial mechanisms and algorithms that emulate the ability of lifelong learning. Each of these methods contribute in a unique way towards approaching an acceptable solution.

Due to the wide variety of approaches taken by previous work, it is difficult to categorize them under a single criterion. However, they may be classified as either reinforced, unsupervised, or supervised learning algorithms, based on the learning rule and type of data provided during training. As the focus of this work is on training agents to interact with their environment, it lies on the boundary between supervised and unsupervised learning. As defined in the background section, reinforcement learning can be seen as the bridge between both these processes. It is a semi-supervised mechanism which allows for an agent to interact with its environment through its senses and allows for the environment to respond with positive/negative feedback. This process is tractable and is modelled after observing the behavior of biological

systems, including ourselves, interacting with the world around them.

2.1.1 Q-Learning

Reinforcement learning has been observed to be a common occurrence in nature, as a mechanism for modelling behavior through positive and negative reinforcement of ideas and actions of the reinforcee deemed desirable to the reinforcer. However, it was first strongly established within the domain of artificial intelligence under the guise of Q-learning [14], which essentially maps real world scenarios into discrete Markovian states. Q-learning further models the state-action values as probabilities of taking a particular discrete action while in a particular discrete state. The rewards received from the environment help promote actions which are more likely to obtain rewards, and thus train the agent. Equation 2.1 indicates how the Q-values are modelled for each state-action pair. The $Q^\pi(s_t, a_t)$ denotes the Q value for state s_t and action a_t . It is modelled as a function of the rewards R received at different times t , given that the current state is s_t and the action being executed or selected is a_t . The discount factor γ is used to reduce the impact of older rewards by weighing down older rewards R_{t+n} and their impact on current Q-values.

$$Q^\pi(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t] \quad (2.1)$$

Q-learning also suffers from several problems, which were overcome in later work, such as the discretization of states, which requires these states to be either predefined or a nature of the environment. For example, Atari games [1] have states predefined, which the agent/player can occupy, or chess where the nature of the environment itself is divided into discrete states as positions on a chess board. However, natural environments lack universally definitive distinguishing features between locales/situations/tasks, and are hence out of the reach of standard Q-learning approaches. Also, the inability to actually learn any logic for switching states other

than maximizing the Q-value for a particular objective, results in the algorithm being unable to adapt to a change in the goal (without having to retrain the entire Q-matrix).

Incorporating Inductive Logic Programming (ILP) into the Q-learning solves the latter problem. This was introduced as Relational Reinforcement Learning (RRL) [15]. ILP implements the decision making process by constructing decision trees and introduces an additional factor "p" which models the relationship between states as P-learning. This factor is binary, in that it simply learns whether the state-action pair is optimal (1) or not (0). This closely resembles the logic followed by decision trees. Although this is not sufficient by itself, combined with Q-learning, the pair of factors are shown to be able to logically tackle simple block based learning environments and tasks in a manner more consistent with logical reasoning than goal-oriented drive. The combination of decisions (P) and learned importance of state-action pairs (Q) results in goal-oriented decision making, and can thus perform even when the target is shifted within the same environment. This is because the P-values are able to attribute a binary reason to the associated Q-values, and thus make the decision making process more conducive. However, this scales poorly to more complex tasks due to the sheer number of variations within the P and Q matrices.

To increase both the speed of learning (and thereby decrease the time to convergence) the RRL approach was extended to support multiple agents [16]. This technique extends the learning capacity of an agent by sharing the learned experiences of multiple agents within an environment. The experiences are collected every cycle (lifetime) and accumulated together after the entire generation expires. This reduces the time to convergence as the knowledge gained by multiple individuals is more likely to be greater and wider than a single individual. Also, simulating multiple entities allows for massive parallelization, which due to the influx of GPUs and cloud computing platforms, is becoming more and more sustainable for individuals

and laboratories. This work also draws on this aspect in one of our test environments, as we spawn multiple agents. Furthermore, multiple agents allow for interactive environments, as the agents not only interact with the environment, but also among themselves. If the ultimate goal to these algorithms is an agent which needs to interact with other people or entities, then a isolated/lone ranger approach would also pose several interactive drawbacks, mainly predictability, and thus leading to engagements being more robotic.

2.1.2 Deep Q-Learning

The stately nature of Q-learning was one of the main hurdles faced by reinforcement learning algorithms of the early 2000s. The seminal paper published by Deepmind [1] overcame this limitation in a manner more consistent with current research. They opted to use a deep neural network in conjunction with the reinforcement learning algorithm (absorbed from Q-learning), in order to remove the dependency of the algorithm on hard-coded states. These networks were most uncreatively named Deep Q-Learning Networks (DQN), and the technique was called Deep Reinforcement Learning (DRL). By mapping the input space to the output actions, the DQN algorithm is able to formulate a relationship between the input and output spaces. The technique also incorporates the time-aspect [14], in a unique manner. This is done by keeping a record of a random number of time-frames within the simulated environment, which treats this record as a mini-batch while learning. This feature allows the algorithm to learn over multiple time-scales. Another limitation of Q-learning, the inability to differentiate between scenarios where the agent is in the same state, but has different history of states was surpassed by this approach. However, the manner of discounting rewards based on their time-based distance from an action, results in agents which are short-sighted in terms of decision making.

The nature journal publication [17] accompanying Deepmind’s research [1], lists

the experiments carried out by DQN. The target for reinforcement learning being several vintage Atari console games. These games although outdated, envision the spirit of learning and highlight agent control. They are task-oriented and diverse in playing style, they have been benchmarked competitively by humans (providing an excellent source of human comparison metrics) and they almost exclusively deal with an agent (sprite/vehicle) in an environment (2D environments only). The benchmarks achieved by Deepmind show DQN outperforming humans on more than 50% of games, and beating other linear algorithms at over 90% of them. This led to DQN being promoted to state of the art in the domain of reinforcement learning, and also established the Atari Games dataset as a standard bench-marking dataset for future work in the field.

OpenAI chose to build upon Deepmind’s work by releasing a collection of Proximal Policy Optimization algorithms (PPO) [8]. The algorithms work in tandem to achieve the latest state-of-the-art results for the Atari Games dataset. They can be divided into 3 separate domains: Policy Optimization, Clipped Surrogate Objective and Adaptive KL Penalties. The learning rule/policy is optimized using policy gradients which estimate the function over a fixed time length and automatically differentiate the gradients to obtain the direction of maximum optimization. They also use Trust Region Methods (TRPO), which bank on the KL-divergence function to penalize the policy for larger divergence values. The values are clipped using a β to limit the effect of the penalty. The work also proposes the use of an adaptive KL divergence method, which essentially just learns/optimizes the β value. This algorithm is released in the public domain, and integrated with our simulation software, thus making it optimum for comparison against the work proposed in this thesis. More recent releases of the PPO algorithms show even more promising results, and do so by ditching the KL-divergence metric in favor of the ratio between old and new probabilities. This surprisingly outperforms the previous implementation and does

so at a lower computational cost.

Another recent offshoot of DRL, Relational Deep Reinforcement Learning [9], claim to use structured perception and relational reasoning to capture the relation of agents in a scene. However, the algorithm simply integrates RRL and DQNs into a single approach. The environments used for testing are more comprehensive and meaningful box worlds (rather than Atari Games). The algorithm constructs underlying graphs to map the relationship between the agent and environment. The process uses a multi-headed dot-product (MHDPA) to compute non-local interactions based on an attention mechanism. The attention mechanism is analogous to the message-passing methodology of graph based networks. This allows the information from different regions to be shared, and thus model a more reason-oriented network. The MHDPA effectively models self-attention (rather than attention to external stimuli), this computes the interaction/relationship between the various inputs (entities being sensed in the environment’s viewport).

2.1.3 NeuroEvolution

Another group of algorithms which progressed alongside Q-Learning and Deep Q-Learning within the field of reinforcement learning, were genetic algorithms. These are a broad category of evolutionary algorithms, namely TWEANNs (Topology and Weight Evolving Neural Networks) focusing on evolving the architecture through various methods in contrast to DRL techniques which focus on training a predefined architecture. Of these, a profound algorithm was proposed in the early 2000s called NEAT or Neuroevolution of Augmenting Topologies [7] which ensembled a variety of biologically (genetically) inspired techniques to match the performance of Q-Learning algorithms. The research paper details many techniques: (i) Genetic encoding, to map a neural network to a Genome and represent it as a Phenotype. (ii) Use of genetic markers to track the genes, and introduction of a novel factor called an innovation

number, which essentially marks the gene with a unique identification code, which is used to identify it during crossover. (iii) Introduction of speciation to protect the innovations in genes from dying out due to any one specific family of genomes dominating the fitness function. (iv) Minimizing the dimensionality of the network by only introducing new nodes when necessary. This "necessity" is very difficult/impossible to gauge and thus the algorithm uses the principles of genetic algorithms to weed out the genomes with added nodes which are under-performing.

Further expanding on the NEAT algorithm, the same group of researchers introduced the HyperNEAT algorithm [18]. Although based on the same base algorithm (NEAT), HyperNEAT is significantly different in both structure and evolutionary function. HyperNEAT is based on CPPNs (Compositional Pattern Producing Networks). These work by mapping a lower dimensional space to a higher dimensional space, with the unique nature of structuring these higher dimensional space with repeating spatial patterns/motifs. The HyperNEAT algorithm encodes these spaces as connections and thus produces spatially symmetric neural networks. It is theorized that these algorithms are better at solving "fractured problems", which involve heightened discontinuity between tasks (example: navigating through a forest), rather than linear continuity (example: race-track racing). As HyperNEAT spatially encodes its nodes, the algorithm allows the network to exploit its own geometry to observe adjacently encoded nodes.

The ES-HyperNEAT algorithm [19] builds directly on top of the HyperNEAT algorithm. This process further optimizes the network structure by dividing the quadtree present in the hypercube's dimensions into a "hypertree" (16 quadrants). The tree nodes are then pruned based on their variance, with the lowest being eliminated. The structure connects all hidden nodes to output nodes without outgoing connections, and all hidden to input connections if there no incoming connection iff the weights of the connections in the hypertree surpass a fixed weight threshold. Thus the ES refers

to the evolvable substrate of the hypercube. The Adaptable ES-HyperNEAT algorithm [20] takes this further by incorporating biologically plausible Hebbian rules to modify the weights. This shows promise for future expansion of this thesis. However, is not applicable to the current implementation. Another work on HyperNEAT tries to apply the algorithm to multi-agent configurations [13]. This uses a "multi-spatial substrate" to essentially separate each input and output into its own independent hyperplane. However, all these implementations are tested on increasingly simplified maze based environments, and have little to no objective than to reach a destination.

2.1.4 Continual Reinforcement Learning

So far, the previous work listed has covered the foundations of reinforcement learning and neuroevolution, however, the problem being tackled by this thesis is life-long/continual learning. The previous work so far has been instrumental in dictating how reinforcement learning algorithms can be used to train agents in environments, however, what needs to be seen is how well these algorithms are able to handle learning multiple different tasks, in different environments, after being exposed to them sequentially. This can be called Continual Reinforcement Learning (CRL).

A first step towards achieving CRL is indicated by a very interesting research algorithm CHILD (Continual, Hierarchical, Incremental Learning and Development) [10] [11]. Here, the proposed algorithm uses a deep reinforcement learning approach (a decade before DRL is proposed), to solve a maze problem. Although the maze itself is a state based environment, the algorithm uses a neural network to solve it rather than a Q-matrix. Furthermore, this algorithm incorporates neurogenesis, by adding new nodes as required. The addition of nodes is done by monitoring the weights of the network and if a particular weight was being pulled in opposing directions, a node was added. This methodology was considerably ahead of its time (for 1994), and was hence only ever tested on very small tasks. Its lifelong learning capabilities

were not able to scale for larger more complex tasks. The method did not have an accounting scheme for multiple time-steps, and instead made an action at every time step. Earlier work by the same author [21] [22] only focuses on the using the same algorithm for multi-layered networks, where it is detailed how incremental layers are added. However, no subsequent work seems to build on this process.

2.1.5 Addressing Catastrophic Forgetting

As stated, one of the hurdles faced by lifelong learning, is retention of previously learned tasks. The phenomenon which causes a network to forget a previously learned task when trained on a new task is termed "Catastrophic Forgetting". Overcoming catastrophic forgetting is targeted by several recent papers striving to tackle the lifelong learning problem. However, apart from the research in CRL mentioned above, the subsequent papers all focus on supervised learning algorithms.

The Elastic Weight Consolidation algorithm (EWC) is amongst the first to undertake the task of overcoming catastrophic forgetting in neural networks in a paper titled exactly as described [4]. This algorithm draws inspiration from the plasticity of synapses, and modulates weights in a manner such that the synapses (weights) responsible for the first task are made less elastic as compared to other weights, thereby allowing the network to encode the second task more thoroughly onto the other nodes of the network. By computing the log loss, and minimizing the penalties for previously learned tasks, the network can effectively learn multiple tasks. This is even shown to work for up to 10 Atari games. However, it is to be noted that the training process does involve interleaving of games. And certain games are trained on multiple times. Furthermore, the computation of the diagonal of the Fisher Information Matrix requires considerable calculations, and thus scales poorly. Thereby limiting this technique to low-dimensional output spaces.

Another algorithm matching the performance of EWC [4] uses "Synaptic Intelli-

gence” [23] to model their algorithm. This is however simply a euphemism to cover intrinsic arithmetic rather than an attempt to draw parallels with any sort of synaptic biology. Here, the algorithm proposes the use of a special importance measure for each weight, which reflects its history. During training, the algorithm retains another factor measuring the amount each parameter contributes towards the loss of a weight. To avoid large modifications to weights more important to prior tasks, the modified cost function proposed adds a ”surrogate loss”, which essentially approximates the summed loss functions of previous tasks.

These algorithms [4] [23] do not effectively demonstrate lifelong learning, and rather demonstrate an efficient method of expanding network capacity. Neither do they show any capacity for learning continually, since they model a fixed size architecture, and thus an eventual saturated network capacity.

Due to computational limitations, many works [10] [11] [12] [13], including this one, have chosen to develop lifelong learning algorithms for intellectually inferior species, such as insects, birds and smaller mammals. This allows researchers the opportunity to develop and test the viability and functioning of an algorithm on a much smaller scale, within the bounds of locally available resources. This also allows for the testing to be carried out in virtual environments [6] [7] [1] [8] [9], which improves evaluation time-frames due to parallelization of scenarios and acceleration of virtual timescales, and lowers the cost of physical setups, environments and hardware equipment.

2.2 DARPA L2M Initiative

DARPA (Defense Advanced Research Projects Agency) recently announced their L2M (Lifelong Learning Machines) initiative in 2017. This work is developed towards attaining the goals set by the program. The program seeks AI and Machine Learning systems which can:

1. Learn continuously
2. Adapt to new tasks
3. Know what to learn
4. Know when to learn
5. Be flexible to different types of task domains

Chapter 3

Related Work

3.1 Reinforcement Learning

Reinforcement Learning is a method of building a relationship between two or more entities based on the feedback received on past interaction between the entities. It is a concept which models behavior through supervision and correction. Although this is similar to the supervised learning paradigm, it differs in the nature of its correction methodology. In supervised learning, the supervisor is aware of the solution (ground truth), however, in reinforcement learning, the supervisor is unaware of the solution, and is only able to comment on whether the action taken was correct or not, and also sometimes the degree to which the action was right or wrong. In a binary problem, this is one and the same, but for more complex problems, this is significantly different. Thus reinforcement learning takes longer to converge to an optimum solution than supervised learning, because the solution remains unknown.

In the context of this work, an agent exploring an environment undergoes reinforcement learning. The agent is the reinforcee and the environment is the supervisor or reinforcer. The agents senses are one set of entities or nodes and the agents actuators (actions/limbs) are the other set of nodes. The relationship between these sets of nodes are unknown. Initially, the relationship is a black-box, and the agent takes seemingly random actions in the environment. The environment in turn pro-

vides feedback to the agent for its actions. For example, if the agent touches fire, he gets burned and the feedback is negative. If he eats food, he becomes energetic and the feedback is positive. The nature of the feedback with knowledge of actions taken helps model the relationship between the agents senses and actions. The feedback is known as the reinforcement signal, the supervisory signal or the reward.

This thesis uses reinforcement learning to teach an agent to survive multiple challenging environments. However, learning multiple tasks consecutively and remembering each of them is out of the scope of standard reinforcement learning. Thus we need to look towards incorporating continual or lifelong learning methods.

3.2 Lifelong Learning

Lifelong Learning is an extended problem to the field of machine intelligence. In the past decade, considerable progress has been made in the fields of deep learning, reinforcement learning etc; the research focuses on problem solving algorithms using these techniques. In recent years, the algorithms have reached human level performance in certain tasks, and other have even surpassed that [1]. However, most algorithms seem to be limited to performing a single task or operate within a single task domain. Learning multiple tasks in a sequential manner, causes the algorithms to fail to remember previously learned tasks. This is known as catastrophic forgetting. And the phenomenon of overcoming catastrophic forgetting and enabling a network to learn continually with minimal loss is called lifelong learning.

As stated in the background section. Several attempts to overcome the limitations of neural networks were made in the hope of achieving lifelong learning [24] [4] [25] [22] [10]. However, algorithmic limitations or computational limitations have prevented significant progress in the area.

Neural networks are a representation of a learning architecture. The amount of information stored by such architectures is limited. An approach to lifelong learning

can be seen as a method of automatically scaling up neural network architectures to fit the information size requirements. However, this leads to explosive increment in memory resources, therefore we also need to keep in mind the amount data needed to be stored. This work addresses this by sharing information between tasks, and only adding memory for new information. The algorithms proposed in the next section, help achieve this. Another restriction is the nature of the learning algorithm itself, which might not be able to capitalize on this scaling ability, but the algorithm proposed in this thesis allows for a continual learning mechanism which accommodates the scalability.

3.3 Neurogenesis, Synaptogenesis and Neuroevolution

As previously stated, increasing the capacity of a neural network can help achieve continual learning by making place for new information. In the context of artificial neural networks (ANN), nodes are representative of connections, and the weights connecting them are representative of the information. The process of adding nodes and weights would be akin to expanding network capacity. In the human brain, neurons represent the nodes. The human brain also has a mechanism for the process of adding neurons, and it is called "Neurogenesis". It is observed both in babies and adults (albeit at different scales).

Furthermore, the process of adding connections (synapses) between neurons is referred to as synaptogenesis. As new nodes are added, synaptogenesis is essential to give the node meaning, as an unconnected node/neuron has no contribution. Thus, in this work, we shall collectively refer to the process of adding neurons and synapses (nodes and edges respectively) under the title of Neurogenesis to prevent the repetition of this term when using the term in a general sense. However, in specific rules and mechanisms for adding nodes and edges, the terms will be used separately.

This work incorporates the use of Neurogenesis into ANNs and with the help of

reinforcement and lifelong learning techniques, it creates a continual learning algorithm. The phenomenon of growing and changing a neural network’s architecture during its learning phase is called neuroevolution. The NEAT algorithm used as a base for this work banks heavily on the process of neuroevolution through an algorithm closely resembling a genetic algorithm. This builds a stable foundation for a lifelong learning model, as genetic evolution (although time consuming) is known to work.

Chapter 4

Algorithmic Specification

4.1 NEAT: Neuroevolution of Augmenting Topologies

The NEAT algorithm [7] developed by Stanley et al in 2002, consolidated several previous techniques into a single comprehensive algorithm, and helped propel the stagnating research in evolutionary computing strategies. The algorithm, as mentioned before, sparked several other works such as HyperNEAT [18], rtNEAT [26] etc [19] [13] [20].

NEAT specifies techniques for neuroevolution through genetic encoding and genetic mutation. These processes occur between the subsequent generation of agent populations. Thus, neuroevolution is dormant during the lifetime of each agent. This methodology has several drawbacks.

Firstly, the experiences and abilities of the agent play no direct role in the evolution process, although, the abilities responsible for survival are indirectly extracted. This leads to much learned information to be wasted between generations. The reduction of redundancy is largely dependent upon the magnitude of the redundancy (redundant features continue to appear if present in a large section of the initial population, and might also reappear after their dissolution, through random mutation).

Secondly, the process by which these abilities are extracted are subject to random chance. Thus the algorithm does not guarantee a convergence or extraction of even

evidently obvious features, and rather banks on a probability distribution over a large population to eventually locate these optimum features.

Finally, the process of neuroevolution itself is also purely based on the random generation of nodes and edges. This process, although more biologically faithful, is inefficient in terms of computation costs, due to the sheer magnitude of simulations required to reach an optimal convergence. It also banks on random chance to discover solutions, which is not a reliable methodology in time-constrained scenarios.

The components of the NEAT algorithm used in this work are listed and specified below:

1. Genetic Encoding

In the original NEAT algorithm, pairs of nodes are encoded as genomes. However, here the running averages and back-propagation toolboxes require all nodes and edges to be in matrix format. So a secondary list of node pairs is maintained. This list is kept out of the agent's code (as it is not required at run-time), the list is only required between generations. Each genome specifies the connecting nodes, their resident layer and the weight of the connection (to prevent memory doubling, the weights are not stored in the list as they can easily be addressed and fetched from the weight matrix). The genome also contains a toggle bit which enables and disables it, and an innovation number which keeps track of it.

2. Mutation

This feature allows the architecture and weight values to change randomly. This allows the network to branch out and behave in an exploratory manner. This random exploration also allows the network to escape a local minima. Since this thesis uses an architecture that follows a layered approach, when a mutation incorporates a node in a new layer, an entire layer is added of size of the immediate lower layer. This is elaborated on in the next subsection 4.2.

3. Crossover

The process of crossover allows genes to be interchanged between members of a population. The aforementioned global innovation number helps keep track of genes. This allows new agents to be created, which share the best features of the previous generation.

4. Speciation

This process is a novelty of the NEAT algorithm over standard genetic algorithm, which allows multiple species to evolve separately. During neuroevolution, the networks may branch out into considerably differing architectures, which may initially differ largely in fitness, thus they can be considered as different species. Speciation helps identify species and maintain crossover only between members of the same species. It also allocates space in the final selection for members of each species by having only compete with other members of the same species. Thus eliminating any particular species from gaining complete dominance.

This thesis incorporates the NEAT algorithm as a baseline, and builds a new algorithm upon this foundation. It mainly uses the NEAT algorithm between generations to explore a larger solution space, while the newly proposed algorithm modifies the network's architecture (at run-time) towards learning the environments.

Speciation is useful when dealing with large changes in architecture, however, this work focuses on evolving the architecture through more direct means, and uses NEAT to a minimum. The population size of each generation is very small (less than or equal to 10), and parameters for mutation are minimal, thus limiting the diversity of evolution through NEAT. Hence speciation is not required, and not implemented in this work.

The next section deals with the novel mechanisms proposed by this thesis, which are implemented atop the NEAT algorithm.

4.2 Relational Neurogenesis

In order to preserve information, learn continually and maintain the size of the network, an algorithm is proposed to incorporate various mechanisms for achieving this purpose. The algorithm is multifaceted and has very specific rules for network growth. As these mechanisms allow a network to grow through neurogenesis based on the relationship between nodes in the network, it is called "Relational Neurogenesis". The algorithm can be seen to have 2 main domains:

1. Network Growth

Mechanisms for Neurogenesis and Synaptogenesis.

2. Preservation of Information

Incorporation of a mechanism for maintaining previously learned information.

The basic operation of the reinforcement learning algorithm can be seen in Figure 4.1. The agents current sensory inputs and past actions are fed in as inputs to a randomly initialized single layered neural network. A single layered neural network is selected as a baseline starting point as it is the minimal requirement for universal function approximation [27], and thus does not impose any disadvantage on the network at initialization. A standard forward pass is conducted which propagates to the output layer. A softmax function at the output layer allows us to calculate and select the highest activated output, and the corresponding action is executed by the agent in the environment. The agent then receives a reward (positive/negative/null) which is used as the error signal for the back-propagation mechanism to update the network's weights accordingly.

After each forward pass, the activations (outputs of each node) and weights are extracted from the network, additionally, the reward and score metric are collected from the environment as well (in blue). These parameters are stored (in red) and are

Figure 4.1: Base Reinforcement Learning Algorithm Operation

used for calculation of various metrics required for Relational Neurogenesis to take place.

4.2.1 Mechanisms for Network Growth

These mechanisms or rules analyze various network parameters either during operation or between episodes. Based on the analysis, a decision is made to modify the existing network architecture. Although there are multiple rules for each step, the process can be categorized as 2 main steps:

- ### 1. Determination of **Need** for Network Growth:

Is neurogenesis required ?

- ## 2. Implementing Network Growth:

Where and How to implement neurogenesis ?

4.2.1.1 Determination of Need for Network Growth:

For a network to grow in proportion with the amount of information it needs to learn, mechanisms must be incorporated to allow this. Network growth can be implemented

through Neurogenesis (adding of nodes) and Synaptogenesis (adding of edges). However, network growth must be curtailed to prevent unnecessary evolution or explosive growth. Thus, the first step in this process is to determine whether network growth is needed at all. To make this decision, several mechanisms are proposed below. The outcome of these mechanisms is to decide whether network growth is required or not.

These mechanisms not only determine whether network growth is required or not, but also which type of network growth is required: Neurogenesis or Synaptogenesis? Similar decisions also need to be made for removing nodes and edges. As different reasons contribute towards different requirements, the processes are delineated below under 4 major categories:

1. Reasons to Add Nodes: Neurogenesis
2. Reasons to Remove Nodes: Neuronal Termination
3. Reasons to Add Edges: Synaptogenesis
4. Reasons to Remove Edges: Synapse Termination

These categories are expressed in figure 4.2 as a continuation of the algorithm in figure 4.1. Figure 4.2 shows the overview of the first step in neurogenesis, which is determining whether it is required or not.

4.2.1.2 Reasons to Add Nodes: Neurogenesis

The first decision is whether new nodes need to be added to a network. This is a big change in the architecture, and cannot be implemented very frequently or it can lead to the formation of very large networks quickly. The following mechanisms are proposed to decide the need for Neurogenesis in this algorithm:

1. Plateauing Merit

Figure 4.2: Relational Neurogenesis Algorithm: PHASE-1

Determining the need for network growth through a series of metrics. Decisions are made to either add or remove nodes, or add or remove edges, based on the various metrics when compared to individual set threshold values.

2. Learning Opposing Concepts

3. Low Margin of Confidence

4.2.1.3 Plateauing Merit

The measure of an agents performance can be viewed as its ability to survive an environment. As such, every environment setup provides a metric of how well the agent is performing within the environment. In the case of Atari Games [1], it is the score (number of points) accumulated over time. In this work, we formulate a similar metric score, which is more naturalistic and can be implemented without the need for a supervisor.

The score is calculated by multiplying the frame count (can be viewed as a discrete unit of time) by the agents health in that frame, as seen in equation 4.1. The score is periodically accumulated over each episode. The longer the agent lives, the higher the score. The healthier the agent remains, the higher the score. The final score at the end of each episode, can be viewed as the Merit Score or the degree of merit achieved by the agent in the episode.

$$Score = Score + (frame \times health) \quad (4.1)$$

As the Merit Score represents the agents performance, the measure of its performance can be done through analysis of said score. If the scores were plotted as a function of the number of episodes, this would indicate the performance improvement or deterioration of the agent. As the agent learns, the merit score would keep increasing. However, if the network architecture limits the agents ability to learn, it would result in either plateauing (constant) merit or declining merit. In either case, network learning capacity can be increased by adding a node.

However, minute score value fluctuations would lead to large addition of nodes if

this criteria is directly applied. Hence, the score is analyzed over an episode window (time window encapsulating a number of episodes) W_{score} . This window stretches from the current episode e_c back to an initial episode e_i as seen in equation 4.2.

$$e_i = e_c - W_{score} \quad (4.2)$$

In order to calculate the change in the merit score, the gradient of the function ΔP is computed at every episode as in equation 4.3.

$$\Delta P = \left[\frac{d(\text{Score}(e))}{de} \right] \approx \frac{(\text{Score}(e_c) - \text{Score}(e_i))}{(e_c - e_i)} \quad (4.3)$$

To prevent instantaneous fluctuations from rapidly adding nodes, the average gradient δP is maintained over the entire episode window as in equation 4.4.

$$\delta P = \frac{\sum_{e=e_i}^{e_i+W_{score}} \Delta P}{W_{score}} \quad (4.4)$$

The final decision to partake in Neurogenesis is decided by comparing this average gradient δP by a set plateau threshold Θ_p . If the average change in gradient is very small or negative, this would imply that the agent is not learning any new information, and would serve as a suitable reason to perform neurogenesis.

$$\delta P < \Theta_p \quad (4.5)$$

It can be noted that plateauing gradients will eventually always be observed when the agent reaches a sufficiently high merit score indicating that there is no new information to be learned. In such a case, neurogenesis will take place unnecessarily. This is an acceptable failure, and the reasoning behind this is manifold. Firstly, if there truly is no more information to be learned and the agent has reached the pinnacle of performance within the network, the nodes added due to this exception will initial-

Figure 4.4: Calculation of plateau gradients for determining need for neurogenesis due to plateauing merit

ize with minuscule weights, and the back-propagation algorithm will not contribute any significant error signal to train the newly weights. Subsequent proposed mechanisms will easily weed out this node and thus maintain equilibrium through very low frequency oscillation of network architecture, which is computationally feasible. Secondly, although eliminating this oscillation entirely would be ideal, the curbing of this functionality would prevent the network from learning newer information presented by the environment in a later stage. In this work, we give priority to the possibility of getting a higher score by temporarily sacrificing performance, but account for the performance in the long run if no higher score is achievable. The operation of plateauing merit can be observed in figure 4.4. The figure, starting from the top-left corner, shows the merit score window, which is then used to obtain the gradient/slope between every subsequent score value. The average of the merit score gradients is then calculated and compared to a threshold Θ_p , which decides whether or not the condition is satisfied to perform neurogenesis.

4.2.1.4 Learning Opposing Concepts

Another mechanism for adding nodes can be derived from the work of Mark Ring [10] [11]. This methodology involves analysis of the weights over time. It is hypothesized, that if the network tries to learn opposing concepts, then the back-propagated error tends to pull certain contributing weights in opposing directions. This is observed in the XOR problem of training a single layered neural network to learn the XOR function. Thus forming another criteria for adding nodes, as this can help solve the problem by offering another node as a degree of separation between the outputs pulling the weights in opposite directions.

A matrix is maintained with the weight change values obtained in every frame after the back-propagation algorithm completes the forward and backward pass. The change in weight $\Delta w(f)$ can be computed as the difference between the current and previous weight (between consecutive frames).

$$\Delta w(f) = w(f) - w(f - 1) \quad (4.6)$$

In order to observe the oscillation in each weight, we compute 2 metrics: The average change in weight $\Delta \bar{w}(f)$ (Eqn 4.7), and the average change in the magnitude of the weight $\Delta \tilde{w}(f)$ (Eqn 4.8), for each weight in the network. The average change is calculated by means of a weighted running average, which gives more recent values more importance as specified by the σ value. Where $0 < \sigma < 1$ and σ closer to 1.

$$\Delta \bar{w}(f) = \sigma \Delta w(f) + (1 - \sigma) \Delta \bar{w}(f - 1) \quad (4.7)$$

$$\Delta \tilde{w}(f) = \sigma |\Delta w(f)| + (1 - \sigma) \Delta \tilde{w}(f - 1) \quad (4.8)$$

If the weight is being pulled in opposing directions, the accumulated average

Figure 4.5: Calculation of weight metrics for determining need for neurogenesis due to the challenging nature of the environment presenting opposing concepts to be learned.

change in magnitude will be considerably higher than accumulated average absolute change, as the absolute change will have positive and negative changes, whereas the magnitude will only have changes in one direction.

If the ratio of these weight metrics is greater than the set weight-pulling threshold Θ_{wp} (Eqn 4.9), then Neurogenesis needs to take place.

$$\Delta \tilde{w}(f) > \Theta_{wp} \cdot \Delta \bar{w}(f) \quad (4.9)$$

By comparing these metrics, we can determine whether the network is trying to learn opposing concepts, and accordingly add neurons to aid it. The process of network growth while learning opposing concepts can be observed in figure 4.5.

4.2.1.5 Low Margin of Confidence

The output layer of the network consists of nodes, each corresponding to an action which the agent can perform. After every forward pass (at each frame), a decision is made to take one action. The output node with the highest activation is selected and the corresponding action is performed.

If a softmax function is appended to the output nodes, the output of the softmax layer can be viewed as the relative confidence of the agent in taking that particular action. And the action with the highest confidence (proportional to the activation) is the one carried out. Thus the softmax is applied to the output layer (Eqn 4.10).

$$confidence(o_i) = co_i = \frac{e^{a(i,L)}}{\sum_{j=0}^{On-1} e^{a(j,L)}} \quad for \ i = 1, \dots, On \quad (4.10)$$

It is ideal that the correct action have the highest confidence, and have the highest margin of difference from the other confidence. As this would correlate to a good degree of assurance that the agent is confident in its decision making process. However, while learning, it is observed that at times the confidence levels of nodes are similar. This indicates that the agent is partially confused as to which action needs to be taken. The degree of similarity between confidence values can be seen as the degree of confusion.

If the output actions are viewed as classes, and the objective is to select one class, then an optimization of this process can be seen as the maximal separation of classes. Thus, in order to maximally separate classes/actions/output nodes, we can add a neuron through neurogenesis so as to provide an additional dimension to facilitate separation between the outputs.

The first step is to obtain the relative confidence $\Delta co_{(i,j)}$ between the output confidence values, so as to measure their similarity (Eqn 4.11). A basic difference serves as a suitable measure. The absolute denotation $|co_i - co_j|$ signifies the irrelevance of

the order of comparison. This metric is computed for all output pairs, except with itself.

$$\Delta co_{(i,j)} = |co_i - co_j| \quad \forall i \neq j \quad (4.11)$$

The relative confidence values are accumulated over f frames within an episode (Eqn 4.12).

$$cco_{(i,j)} = cco_{(i,j)} + \Delta co_{(i,j)} \quad (4.12)$$

It can be noted that as the relative confidence values are reversible, they do not need to be computed twice. Thus instead of an n^2 problem, it is only an $\frac{n \cdot (n-1)}{2}$ computationally complex problem.

$$\Delta co_{(i,j)} = \Delta co_{(j,i)} \quad (4.13)$$

The average relative confidence value $\bar{co}_{(i,j)}$ of each confidence pair is computed to measure the similarity over time (Eqn 4.14).

$$\bar{co}_{(i,j)} = \frac{cco_{(i,j)}}{f} \quad (4.14)$$

The average $\bar{co}_{(i,j)}$ of the entire output layer $\bar{col}$ is also calculated and used as a relative reference for comparison, rather than an absolute reference.

$$\bar{col} = \frac{\sum_{i=0}^{On-1} \left[\sum_{j=0}^{On-1} \bar{co}_{(i,j)} \right]}{2 \cdot On} \quad (4.15)$$

Finally, the average relative confidence for each pair is compared to the collective average of the entire output layer, and if the ratio is greater than a set threshold value, then it would indicate that the pair with very high correlation of output confidences

Figure 4.6: Calculation of weight metrics for determining need for neurogenesis due to a low margin of difference between output confidence values.

over time need to be separated.

$$\bar{co}_{(i,j)} < \Theta_{cc} \cdot \bar{col} \quad (4.16)$$

In order to perform that separation, the decision is made to add neurons through neurogenesis if this condition (Eqn 4.16) is satisfied. The process of network growth due to a low margin of difference between output confidence values can be observed in figure 4.6.

4.2.1.6 Reasons to Remove Nodes: Neuronal Termination

The second decision is whether any nodes need to be removed from a network. Removing nodes in a lifelong learning network is a dangerous task. This is because, an analysis of network activity in the current task does not reflect the importance of neu-

rons in previously learned. Thus, the objective of removing nodes from such networks can be viewed as a hurdle. This work deduces only 2 safe mechanisms for removing nodes while preserving previously learned information. The following mechanisms are proposed to decide the need for Neuronal Termination in this algorithm:

1. Node Not Contributing
2. Identical Activation

4.2.1.7 Node Not Contributing

One of the primary reasons to remove nodes are if they are not contributing towards the task solving process. This can be ascertained by measuring the activity of the nodes over time. The nodes with poor activation can be easily weeded out. However, in the context of lifelong learning networks, this can lead directly to catastrophic failure on previously learned tasks. The current activations being analyzed would not reflect activation to previous tasks, and nodes important to previous tasks, which would appear seemingly inactivate could be removed, thus causing catastrophic forgetting, and maybe even network failure due to node interdependence.

Thus we propose to remove nodes, if the weights connected to it are on average quite low. Thus the product of the weights with incoming activations would also be low.

However, how does one define low? In this case, we can compare the average weight of each node to the other nodes to analyze its relative importance with respect to the other nodes in the network.

First, we need to compute the global average weight of every node $\bar{w}_{global}$ (Eqn 4.17).

$$\bar{w}_{global} = \frac{\sum_{i=0}^{W_n-1} |w_i|}{W_n} \quad (4.17)$$

Then we compute the average weight of each individual node $\bar{w}_{local}$ (Eqn 4.18).

$$\bar{w}_{local} = \frac{\sum_{i=0}^{Wn_{nl}-1} |w_i|}{Wn_{nl}} \quad (4.18)$$

The comparison of these averages conveys the relative importance of that node to other nodes in the network. This is because the activation of every node is directly proportional to the weights connected to it.

$$\bar{w}_{local} < \Theta_{lnew} \cdot w_{global} \quad (4.19)$$

If the ratio between these averages is too small as compared to a set Local Node Elimination Weight Threshold threshold Θ_{lnew} , then the decision can be made to perform neuronal termination, wherein the node satisfying the equation (Eqn 4.19) is removed.

4.2.1.8 Identical Activation

Another reason for removing can be the presence of duplicates. Nodes with the same weights connected to the same nodes can be seen as duplicates, however, this is relatively straightforward and fairly computationally tedious to track, as we would need to compare every node's weight pair with every other nodes weight pair and also the weight values. In need for a more comprehensive measure of the similarity between nodes, we turn to the similarity measure between the activity of hidden neurons.

The logic being, that if 2 nodes in a layer are active at the same time and to the same degree, over an entire episode, then one of them is redundant. The obvious flaw to this logic is that, if the inputs contributing to the nodes are different then the reasons for the activation are different. However, this does not counter the redundancy of the node, it simply explains a reason for its validity. If the objective of the algorithm

is to tackle fault tolerance through redundancies, then this would be a valid reason to preserve such nodes. However, as the objective of this algorithm is to maintain network growth, redundancies might need to be pruned to limit network growth. (Fault tolerance in this algorithm is tackled through recovery instead of redundancy).

In a process similar to 4.2.1.5, we calculate the relative activation $\Delta ra_{(n_1, n_2, l)}$ between nodes in a layer, for all node pairs within the layer (Eqn 4.20).

$$\Delta ra_{(n_1, n_2, l)} = |a_{(n_1, l)} - a_{(n_2, l)}| \quad \forall n_1 \neq n_2 \quad (4.20)$$

The relative activation values are then accumulated over f frames within an episode (Eqn 4.21).

$$ra_{(n_1, n_2, l)} = ra_{(n_1, n_2, l)} + \Delta ra_{(n_1, n_2, l)} \quad (4.21)$$

It can be noted that as the relative activation values are reversible, they do not need to be computed twice. Thus instead of an n^2 problem, it is also a $\frac{n \cdot (n-1)}{2}$ computationally complex problem.

$$\Delta ra_{(n_1, n_2, l)} = \Delta ra_{(n_2, n_1, l)} \quad (4.22)$$

The average relative activation value $\bar{ra}_{(n_1, n_2, l)}$ of each hidden node pair within a layer is computed to measure their similarity over time (Eqn 4.23).

$$\bar{ra}_{(n_1, n_2, l)} = \frac{ra_{(n_1, n_2, l)}}{f} \quad (4.23)$$

The average $\bar{ra}_{(n_1, n_2, l)}$ for each hidden layer $\bar{lra}_l$ is also calculated and used as a relative reference for comparison, rather than an absolute reference (Eqn 4.24).

$$\bar{lra}_l = \frac{\sum_{n_1=0}^{TN_l-1} \left[\sum_{n_2=0}^{TN_l-1} \bar{ra}_{(n_1, n_2, l)} \right]}{2 \cdot TN_l} \quad (4.24)$$

Finally, the average relative activation for each pair is compared to the collective average of the entire hidden layer, and if the ratio is lesser than a set threshold value, then it would indicate that the pair with very high activity correlation are almost identical or identical.

$$\bar{r}a_{(n_1, n_2, l)} < \Theta_{ia} \cdot \bar{l}ra_l \quad (4.25)$$

As discussed, identical nodes introduce redundancy, and in order reduce the network size, the decision is made to remove one of the nodes (from the redundant pair) if this condition (Eqn 4.25) is satisfied.

Finally, it is necessary to tie up loose ends, and note that while the inputs to the redundant (removed) node are terminated, the output weights are significant and need to be preserved. The output edges of the removed neuron are connected to the survivor node with the weights preserved.

4.2.1.9 Reasons to Add Edges: Synaptogenesis

The third decision is whether any edges need to be added to a network. This work proposes 2 mechanisms for adding edges. Adding edges to a network is akin to synaptogenesis (formation of synapses). The following mechanisms are proposed to determine the need for Synaptogenesis in this algorithm:

1. Random Exploration
2. Poor Connectivity between Layers

4.2.1.10 Random Exploration

One of the key aspects of reinforcement learning agents is they ability to explore unknown terrain and build relationships between their senses and their actions to best adapt to their environments. However, simply learning from the environment

leads to the agent converging to the nearest local minima. However, as virtual agents do not experience boredom or curiosity, they are content with the local minima. In order to escape these wells, the agent needs to emulate curiosity, and this is done through random exploration.

$$random(0, 1) < \Theta_{ex} \quad (4.26)$$

A small exploration factor Θ_{ex} is selected, and if by random chance the condition in Eqn 4.26 is satisfied, then a random edge is added to the network. This is done to explore new architectures so as to escape local gradient wells.

4.2.1.11 Poor Connectivity between Layers

As inputs propagate forward through the network, nodes in every subsequent layer get activated. As the networks are sparsely initialized, there are chances that the initialization is insufficient for activations to propagate satisfactorily through the layers of the network. This can be viewed as poor connectivity between layers, and can be solved by adding edges to the nodes between poorly connected layers.

To determine the degree of connectivity, we monitor the activity of the layers with respect to the activity of the network as a whole.

First, the average activity of the entire network AF_G is determined by averaging the activities of all the nodes at a given time frame (Eqn 4.27)

$$AF_G = \frac{1}{(L)} \cdot \sum_{l=0}^{L-1} \left[\frac{\sum_{n=0}^{TN_l-1} a_{(l,n)}}{TN_l} \right] \quad (4.27)$$

Then, the average layer-wise activity of the entire network AF_{LW} is determined by averaging the activities of all the nodes in a layer at a given time frame. This metric however, weighs layers' activities according to their depth within the network, where layers closer to the output are more important as compared to those closer

to the input. This is because, the activity propagates from the input to the output, hence there is a greater likelihood of layers closer to the input to be more activated than later layers. Thus the deeper layers are weighted more heavily (Eqn 4.28).

$$AF_{LW} = \frac{2}{(L+1)} \cdot \left[\sum_{l=0}^{L-1} \frac{(l+1)}{L} \left[\frac{\sum_{n=0}^{TN_l-1} a_{(l,n)}}{TN_l} \right] \right] \quad (4.28)$$

If the layer-wise average is significantly smaller than the global average, then it can be deduced that the later network on an average has poor propagation of activations from input to the output and thus exhibit poor connectivity of hidden layers.

$$AF_{LW} < \Theta_{ma} \cdot AF_G \quad (4.29)$$

In order to remedy the connectivity issue, synaptogenesis is implemented given that the condition stated in Eqn 4.29 is satisfied.

4.2.1.12 Reasons to Remove Edges: Terminating Synapses

The fourth decision is whether any edges need to be removed from the network. Removing edges from reinforcement learning networks can be seen as attempts to prune the network size. However, in lifelong learning networks, this must be done extremely carefully because information preserved in these edges could pertain to previously learned tasks, which could lead to catastrophic forgetting. Edges in a network are similar to synapses in the brain. The following mechanisms are the only safe deduced mechanisms for terminating synapses from the network.

1. Weight Not Contributing
2. Removal of Nodes

4.2.1.13 Weight Not Contributing

If an edge weight is negligible in magnitude, then the product of that weight and the activation will also be negligible. As the impact of that edge can be seen as minimally valuable to all tasks, the removal of that edge can be seen as a valid step to prune the network, without any significant loss of information.

$$|w| < \Theta_{gwe} \cdot w_{global} \quad (4.30)$$

4.2.1.14 Removal of Nodes

If a node is removed, then the corresponding edges are also removed. Precautions are taken to remove nodes very carefully, and contributing nodes are terminated only if they are determined to not be significantly useful to the networks performance. Thus, this is a safe operation.

4.2.1.15 Implementing Network Growth

In the previous section, various mechanisms were proposed and explained for determining whether a network was in need of neurogenesis or synaptogenesis. These mechanisms, especially for adding architectural elements, analyze network parameters and accordingly determine whether a network needs a node or an edge, however they do not actually perform these tasks. In this section, we explore the rules for neurogenesis and synaptogenesis, once a decision has been made to implement them.

$$[A_I] = \begin{bmatrix} a_{0,0} & a_{0,2} & \dots & a_{0,TN_0} \\ a_{1,1} & a_{1,2} & \dots & a_{1,TN_1} \\ \vdots & \vdots & & \vdots \\ a_{L,0} & a_{L,2} & \dots & a_{L,TN_l} \end{bmatrix} \quad (4.31)$$

$$[A_R] = [A_R] + [A_I] \cdot R \quad (4.32)$$

- Where to Add Nodes:

1. For Plateauing Merit

Layer with the Lowest Average Relative Activation

2. For Low Margin of Confidence

Layer with the Lowest Average Relative Activation

3. For Opposing Concepts

Lower Layer directly connected to the weight satisfying the the weight-pulling condition by the largest margin

- Where to Add Edges:

1. For Newly Added Nodes

- (a) If Node is in the lowest (first) layer

Fully Connect to all Inputs

- (b) If Node is in the last (hidden) layer

Connect Node to the Output pair which has the smallest Average Relative Confidence between Outputs

- (c) For Other Internal Connections

Use criteria for Existing Nodes

2. For Existing Nodes

Connect to highest positive $[A_I]$ value in layer below and/or above.

- How to Initialize Weights:

1. $\text{Random}(0,1) \cdot \mu_{wsi}$

4.2.2 Mechanism for Preservation of Information

As described in previous sections, an important feature of lifelong learning algorithms, which is key to "lifelong learning" itself, is the ability to overcome catastrophic forgetting. This involves identifying sections/parts of the network which have information important to the tasks-to-be-remembered and preserving the information in those sections.

Both LWF [28] and EWC [4] algorithms preserve information by appropriately measuring the entropy encoded by each task into each weight, and preserving it proportionally. However, both these techniques and other techniques [29] [25] [30] require a context provider, which is a supervisory entity (in these cases, the researchers themselves) which need to provide explicit context to the algorithm while switching between tasks. This results in loss of autonomy in exploratory lifelong learning agents. This thesis proposes 2 mechanisms, drawing from the methodology of previous work, in order to preserve learned information and facilitate lifelong learning without explicit context being provided.

Primarily, the preservation of learned weights is done by limiting the amount of modification done to the weights during back-propagation. This is carried out by assigning a different learning rate to each node in proportion to its relevance to a task based on its position in the sequence of learning environments.

As a node is added to the network, a learning rate is initialized to it. All new nodes are given the same learning rate upon initialization. When a new task is introduced (task/context shift), the learning rate of all nodes in the network is scaled down by a given scaling factor in proportion to their activity in the previous episodes. The learning rates are maintained in a learning rate matrix $[LR]$. Eqn 4.33 shows the calculation of the new learning rate matrix based on the Hadamard product of old learning rate and the Hadamard inverse of the product of the modulus of the residual

activation matrix $[A_R]$ and the scaling factor λ_{LR} .

$$[LR] = [LR] \circ [|A_I| \cdot \lambda_{LR}]^{\circ-1} \quad (4.33)$$

The more active a node is during an episode indicates the importance to the task being performed during that episode. Thus, more activated nodes are appropriately preserved by having their learning rates greatly decremented. This operation is only performed when a context switch or a change of task/environment is detected. However, as there is an absence of a context provider, a secondary mechanism is proposed to automatically detect context shifting.

The secondary mechanism operates by analyzing the instantaneous changes in the merit score. If the environment/task is changed, and the agent is suddenly unable to cope with its new surroundings, this instability will be reflected in the agent's score. The score is expected to drop by a considerable amount. However, this is also observed when the agent starts learning any environment (there are continuous score fluctuations), thus we compare the instantaneous gradient of the score value ΔP with the average gradient of the score value δP .

$$\Delta P > \lambda_{SS} \cdot (1 + \delta P) \quad (4.34)$$

Eqn 4.34 shows the relation between the gradient metrics, satisfying which the algorithm can understand that the task/environment has been changed and that the previously learned information needs to be preserved. λ_{SS} is the score sensitivity factor, which determines how much the score needs to drop with respect to the plateau in order for a context shift to be detected.

Parameter	Description
f	Simulation Frame (Periodic Time)
e	Simulation Episode (epoch)
l	Layer Number
L	Total Number of Layers
AN_l	Total Number of Activated Nodes in Layer l
TN_l	Total Number of Nodes in Layer l
On	Total Number of Outputs
In	Total Number of Inputs
Wn	Total Number of Weights (Edges)
$Score(e)$	Final Score in Episode e
W_{score}	Window for Score Plateauing
W_{weight}	Window for Weight Pulling
ΔP	Plateau Gradient
δP	Average Plateau Gradient
w	Weights of Edges connecting Nodes
Δw	Change in Weight
$\Delta \bar{w}$	Average Change in Weight
$\Delta \tilde{w}$	Average Change in Weight Magnitude
co_i	Confidence of i^{th} output
$\Delta co_{i,j}$	Relative Confidence between Outputs i and j
$\bar{co}_{i,j}$	Average Relative Confidence between Outputs i and j
$\bar{col}$	Average Relative Output Layer Confidence
$a_{(n,l)}$	Activation value of Nodes n in Layer l
$\Delta ra_{(n_1,n_2,l)}$	Relative Activation between Nodes n_1 and n_2 of layer l
$\bar{ra}_{(n_1,n_2,l)}$	Average Relative Activation between Nodes n_1 and n_2 of layer l
$\bar{lr}a_l$	Average Relative Activation of Layer l
AF_{LW}	Layerwise Weighted Global Activity Factor
AF_G	Unweighted Global Activity Factor
Θ_{wp}	Weight Pulling Threshold
Θ_{gwe}	Global Weight Elimination Threshold
Θ_{lnew}	Local Node Elimination Weight Threshold
Θ_{ia}	Identical Activation Threshold
Θ_{ma}	Minimal Activity Threshold
Θ_{cc}	Confidence Correlation Threshold
δ_t	Compatibility Threshold
λ_{LR}	Learning Rate Scaling Factor
μ_{wsi}	Weight Scaling Initialization Factor
$[A_I]$	Input Activation Matrix
$[A_R]$	Residual Activation Matrix
$[LR]$	Learning Rate Matrix
R	Reward from Environment

Table 4.1: User-Defined Parameters for Relational Neurogenesis

Chapter 5

Test Environments

A comprehensive simulation environment is key towards an algorithm being effective in the real world. The various facets supporting such an environment depend greatly on the nature and capabilities of a software framework. In this chapter, the simulation software is explained, along with its features. And the various test environments running in the simulation software are detailed and justified. Lastly, the communication protocol is also explained to support the architecture.

5.1 Unity Engine

The simulation environment described in this section and referred to in the entirety of this dissertation for the testing and training of the proposed algorithm is the Unity Engine [31]. (Also referred to as Unity3D and Unity Game Development software). It is a 2D and 3D game design integrated development environment (IDE), which is free for all non-commercial use, including academic research. It also supports a backend runtime engine, known as the Unity Engine. This software, in the context of this work, allows for the development and design of virtual simulation environments. These virtual environments can be designed as required to emulate natural, artificial and hypothetical environments. The Unity Engine itself has an inbuilt physics simulation model which can render rigid and soft body physics between entities and the environment in real-time.

As of the 2018 release, Unity can render high-quality photo-realistic models, nearly indistinguishable from real objects. Although the physics can never capture the subtle intricacies of particulate/molecular/chemical physics, it can make a relatively close approximation on scales similar to the entities dimensions. Furthermore, complex natural phenomena such as wind, fire, gasses, smoke, dust, friction, magnetism, refraction, lighting, shadows, sounds, odors and vibrations can all be simulated in real-time.

As a game environment, it supports the spawning of characters/agents/entities into the environment as required. The C# backend scripting allows for communication with other machine learning scripts to facilitate any external AI control mechanism.

5.2 Description of Environments

The following subsections describe each environment in detail. Furthermore, the design choices for each environment are justified. The environments are all evaluated based on the survival duration and objective achieving efficiency of the agent. The agents selected for this dissertation are a spider (environment set A) and a drone (environment set B).

5.2.1 Environment Set - A: Spider Survival

The spider emulates a low-intelligence creature, and this work has pre-programmed the ability to walk into the agent. The spider is selected due to the nature of its 4-8 compound eyes. These eyes have poor clarity and quality of vision is limited to rapid detection of moving objects due to changes in light and dark areas. They have a keen sense of smell, and can sense subtle changes in temperature. However, they have low tolerance for temperature changes and become inactive (unable to function) in the cold. They are also tremendously sensitive to vibrations. These senses are emulated

Figure 5.1: Sensory Perception Zones for Auditory, Vibratory, Olfactory and Temperature based senses. The degree of sensory impact is divided into 3 zones, which scale up the sensory input by a factor of 10 per zone as you come closer to the spider.

in the digital spider with varying degrees of impact as shown in figure 5.1.

The spiders agent here is allocated 5 eyes. These eyes act like straight line distance sensors, but they can also sample the color/intensity of the entity at the end of the line of sight (LOS). The visual system is shown in figure 5.2 These eyes are aligned such that 1 faces directly ahead, 2 face directly towards the sides, and another 2 are angled at 45 degrees to every other eye, such that one points front-left, and the other front-right. These factors are accurately modelled into the spiders sense-based agent (Not biologically accurate, but accurate relative to the environment) and faithfully reproduced and tuned to the required sensitivity of the virtual environment.

Each environment is based on a randomly generated maze, the maze is randomly generated at the start of training, and remains the same for the entirety of the training and testing session. Also, the mazes are the same for each environment within the environment set, so as to encourage the spider to learn the environment around him, enabling him to navigate more efficiently after training. The spider can sense the maze through its eyes. The maze appears green in color to the spider, in order to

Figure 5.2: Spider Visual System, which highlights the 5 LOS trajectories which the spider can see at one time.

differentiate it from other entities. Small negative rewards are given for coming into contact with the maze boundaries or the maze wall itself. The spider has 4 actuators or actions, each having a continuous value. The spider can rotate (turn) left or right, or translate (move/walk) ahead or behind. The reasons for each sense are described as required by each environment listed below.

5.2.1.1 Environment - A1: Food

The first environment is a maze. Food is intermittently spawned at random intervals of time, at random places in the maze. The spider can sense the food through the emulated sense of smell. The smell is a continuous 8-bit value, which is modelled to be at an intensity level inversely proportional to the square of the distance of the spider and the food. The spider can also see the food if in LOS of its 5 eyes. Food appears blue to it. The spider has a health level which continuously decreases linearly with respect to time. If the spider comes into contact with the food, its health is restored by a fixed discrete value. The inverse square proportionality allows the network to be tested on its ability to model non-linear functions, and the linear drop in health allows the network to be tested on its ability to model linear functions. The first

Figure 5.3: Environment - A1: Food. Cockroaches serve as food, and the spider must locate the dead cockroaches in a timely manner in order to obtain a commensurate reward.

environment tests the ability of the spider to accomplish a very primordial task of food gathering in order to survive. Should the spider's health ever drop to zero, the run is considered finished. Negative rewards are given for inaction. No rewards are given for movement, but health drops quicker over time when the spider is moving. Positive rewards are awarded for getting the food. Thus the environment is called "Food".

5.2.1.2 Environment - A2: Fire

The second environment is also a maze, modelled to the exact specifications of environment A1. Fire spawns at random intervals of time, at random places in the maze. Here the spider can sense the temperature, as a modelled sense of sensation. This sensation is a continuous 8-bit value, which is modelled to be at an intensity level inversely proportional to the square of the distance between the spider and the fire. The spider can also see the fire if in LOS of its 5 eyes. Fire appears red to it. The spider has a health level which continuously decreases non-linearly with respect to time. If the spider comes into contact with the fire, its health is decremented by a

Figure 5.4: Environment - A2: Fire. Flames spawn at random points within the maze. The spider must identify these regions and remain close to them, however, must learn not to actually touch the flames.

large discrete value. Also, the rate at which the spider's health drops with respect to time is directly proportional to the square of its distance from the flame. The rate ranges between -1 and +1, thus allowing the spider to regain its health if within warming distance from a flame. This allows the network to be tested on its ability to model multiple conflicting non-linear functions. The second environment tests the ability of the spider to accomplish a fundamental task of keeping warm while still understanding the dangers of a flame. Should the spider's health ever drop to zero, the run is considered finished. Negative rewards are given for inaction. No rewards are given for movement, but health drops quicker over time when the spider is moving. Positive rewards are awarded for staying warm and close to the fire (but not too close). Thus the environment is called "Fire".

5.2.1.3 Environment - A3: Foe

The third environment is also a maze, modelled to the exact specifications of environments A1 and A2. An enemy agent spawns at random intervals of time, at random

places in the maze, but not within a minimal radius of the spider. The enemy or foe moves randomly throughout the maze, without knowledge of where the spider is. It too is modelled with 5 LOS eyes. However, should the spider come into the LOS of any of its eyes, then the foe will pursue the spider. This pursuit is continued until the LOS is lost, after the enemy has reached the last known position of the spider. The spider is given a benefit of being slightly faster than the enemy, but has the disadvantage of having limited health (which decreases as it moves). Here the spider can sense the vibrations, as a modelled sense of sensation. This sensation is a continuous 8-bit value, which is modelled to be at an intensity level inversely proportional to the square of the distance between the spider and the enemy. The spider can also see the enemy if in LOS of its 5 eyes. The enemy appears black to it. The spider has a health level which increases with respect to time, if the spider is stationary. However, if in motion, the health drops proportional to the amount of translation (movement). If the spider comes into contact with the enemy, its health is decremented by a large discrete value. This allows the network to be tested on its ability to model binary and linear functions. The third environment tests the ability of the spider to stay safe in the role of a prey, and yet conserve energy. Should the spider's health ever drop to zero, the run is considered finished. Negative rewards are given for movement. Positive rewards are awarded for staying still and conserving energy. Negative rewards are given for coming into direct contact (collision) with the foe. Thus the environment is called "Foe".

5.2.1.4 Environment - A4: Friend

The fourth environment is also a maze, modelled to the exact specifications of environments A1, A2 and A3. Here, another spider spawns at the same time as our primary agent. Both agents learn according to the algorithm specified. But both networks operate independently, like separate entities. However, only the primary spider

Figure 5.5: Environment - A3: Foe. A predatory spider will try to eat the host spider. The spider must learn to stay out of its LOS by identifying the best paths in the map, any safe zones or learn to stalk the predator from behind (it’s blind side).

is under observation for testing and evaluation. Both spiders are identical (colors are different in the simulation environment for our benefit, but they appear identical to each other). The spider can see and feel vibrations of the other spider, much like an enemy. The spiders appear white in color to each other. No rewards for movement. Negative rewards for inaction. Negative rewards are awarded for contact between spiders. Positive rewards for prolonged spatial association between spiders (when they are close together). This encourages the spider to make friends and benefit from each others company, but not engage directly. Thus the environment is called "Friend".

5.2.1.5 Combined Environments

Although the agent is trained on the environments 1 through 4, in order to evaluate the algorithm for lifelong learning and specialized learning, we combine environments in various combinations to evaluate different learning strategies and algorithmic capabilities. The spider’s behaviour is also under scrutiny to see what develops without any specific directive being given. The different combined environments are as listed

Figure 5.6: Environment - A4: Friend. Another host AI agent (another spider) will spawn alongside the primary agent. Although only 1 of the 2 will be evaluated, both will operate under the same criteria and be controlled by the same algorithm. The goal being to observe the collective survival behavior of these agents without explicit directives.

below.

1. Food and Fire

Agent learns to go towards something and stay away from something else.

2. Food, Fire and Foe

Agent learns to maintain health and warmth, but stay away from enemies.

3. Food, Fire and Friend

Agent learns to maintain health and warmth, and make friends. However, we evaluate whether the spider shares food or fights over it.

4. Food, Fire, Friend and Foe

Agent learns to maintain health and warmth, and make friends. However, we evaluate whether the spider shares food or fights over it, and whether they work together against enemies or not. The spiders are motivated to gather food

alone, so as to not share resources, but together, the enemies attention is split between spiders.

5.2.2 Environment Set - B: Forest Fire Rescue

This environment set is centered around a drone (more specifically, a quadcopter) as the primary agent. Inspired by the growing number of wildfires and forest fires taking place world over, this environment recreates a similar situation within the unity engine. The ultimate goal being to assist in rescue operations for people trapped in these situations. Thus the environment is modelled to generate a random forest distribution with several tree clusters as seen in figure 5.7.

The drone is equipped with several sensors. On-board thermal sensors measure its internal and external temperature encoded as 8-bit values between 0-255. An FLIR camera (Forward-looking infrared) is equipped to obtain a 16x9 image heat-map of its immediate line of sight. Each pixel obtains an 8-bit value proportional to the average heat given off by the region of the environment focused onto that pixel. The drone is also equipped with a standard RGB camera with a resolution down-scaled to 16x9 pixels to identify the objects in the scene. A pre-trained network identifies different objects in the scene (this is not part of this algorithm). It can only differentiate between trees, fire, the ground, people and fire-trucks.

The drone is equipped with a GPS sensor, but to make the scenario more realistic, we intermittently turn off the GPS sensor. The GPS availability is set at 50%, which means that at every input frame there will be a 50% probability of GPS signal dropping out. Another input senses whether the GPS is operational or nonoperational to eliminate the ambiguity of null inputs. The final sensor is an auditory (microphone) sensor which is tuned to pickup human voices and fire-engine sirens. The auditory network is not a part of this work, and is instead hard-coded into the simulation. Lastly there exists a signalling beacon to indicate whether a person has been located

Figure 5.7: A sample forest environment. Generated terrain and trees. Trees are distributed by clusters, most of which will be set on fire as designated as hot-zones.

Agent Drone

Figure 5.8: Simulated Drone system with some sensors. This is the agent used in environment set B (Forest Fire). The major sensor systems are highlighted in the diagram.

or not. It is mounted on the drone's back.

The final environment also requires additional inputs from other drones signalling in. This is done by keeping a 2 inputs from external sources. These are for X and Y GPS coordinate values from incoming signalling drones.

5.2.2.1 Environment - B1: Fire

The first environment is a forest, with several dense tree clusters. Several of which are engulfed in flames. The objective of this environment is to navigate through the forest covering maximal area without overheating, crashing or coming into contact with flames. The drone is equipped with thermal sensors and other electronic equipment. Thus should learn to navigate the forest without crashing into the ground or the trees. The drones electronic and mechanical equipment is sensitive to extreme temperatures and overheating and thus must learn to balance its own temperature while operating. This is done by reducing its time in hot-zones. The transfer of heat is proportional to its distance from the flames. Direct contact with flames is severely damaging. Overheating can lead to drone failure, and ending of the simulation. However, in

RGB Camera (HD)

Processed Image (16x9)

FLIR Camera (16x9)

Object Detection from Visible and IR Spectra

Figure 5.9: RAW frames captured from the drone shows data from the FLIR camera as well as the RGB camera. The cumulative effect covers both visible and IR spectra and thus provides more robust human detection. The human body temperature range is very stable and thus easier to locate using the FLIR camera. It falls comfortably between the temperature of ground/trees and the temperature of the flames.

Figure 5.10: Environment - B1: Fire. The drone agent explores the forest during a forest fire with the goal to cover as much area as possible in the shortest possible time. The drone must learn to avoid overheating by balancing its time between hot and cool zones.

order to motivate the drone to move, the drones coverage of forest area is assessed as the score. This indicates the drones ability to cover ground maximally.

Positive rewards are awarded for covering more ground and keeping temperature above a safe threshold. Negative rewards for overheating, coming in contact with fire, or remaining idle. Battery life of the drone is also factored into the equation.

5.2.2.2 Environment - B2: Alert

The second environment is the same forest as in environment B1. However, this time there is no fire. The objective is to locate and identify humans trapped in the forest by following their vocal cries and using the on-board FLIR and RGB cameras. The drone's microphone sensor will detect human voices, and the drone must learn to navigate towards them. Upon identifying people, the drone must turn on its beacon to relay the location data to fire marshals and rescue operatives.

Positive rewards are given for covering most ground and locating people. Negative rewards are given for idle flight. Battery life of the drone is also factored into the equation.

Figure 5.11: Environment - B2: Alert. The drone agent explores the forest in search of lost/trapped people. It is seen here identifying the people in front of it. It is rewarded for every person that it finds.

Figure 5.12: Environment - B3: Rescue. The drone is seen rescuing 2 people by back tracking and leading them out of the fire and to the nearest fire-engine.

5.2.2.3 Environment - B3: Rescue

The third environment is the same forest as in environments B1 and B2. The objective of this environment is to not only locate the people trapped in the forest but also lead them to a fire-engine. This is an extension of combined environments B1 and B2, as the forest is on fire this time unlike in B2. Additionally, unlike B2, the drone must lead the people to the fire engine upon locating them. The drone must locate the nearest fire-engines either through their siren or through the visual systems. The drones will move slower while escorting people out.

Positive rewards are given for covering most ground, locating people and rescuing

Figure 5.13: Environment - B4: Team. As each drone is limited to rescuing only 2 humans at a time, this environment explores the use of 4 agents. Upon locating large groups of people, the drone signals other drones to assist by relaying the GPS location information to them.

people. Negative rewards are given for idle flight, and overheating. Battery life of the drone is also factored into the equation.

5.2.2.4 Environment - B4: Team

The fourth environment is the same forest as in environments B1, B2 and B3. However, it differs in the manner that instead of a single drone, there are 4 drones. They all spawn at the same time and are identical in every way. All drone agents learn according to the same algorithm specified. But their networks operate independently, like separate entities. However, unlike environment set A, the cumulative score for all drones is evaluated together. The objective of this scenario is to assess the ability of the drones to learn to operate together. One wishes to observe the drones signalling other drones to collectively rescue large groups of people as they sometimes tend to be clustered together in such situations.

5.2.2.5 Combined Environments

The combined environments section is omitted from this environment set, as the environments are already heavily combined. As seen in sections 5.2.2.3 and 5.2.2.4.

5.3 Communication Protocol Library

This thesis also proposes a novel communication library for communicating between the Unity Engine [31] and any external machine learning tools. Prior to this work, there are several implementations of reinforcement learning environments, the most popular of which is the OpenAI gym framework [32]. This suffers from several drawbacks such as:

1. The limitation of the framework's library and functionality.
2. The restriction of using the python language only.
3. The limited control over the environments.
4. Difficulty and inconvenience of setting up your own environment.

The Unity Engine allows users to develop custom learning environments with high fidelity, accuracy and quality of content. The tools are easy to learn and use, and are free. Furthermore, Unity Engine has libraries for integration with Tensorflow (python and C#) using the UnityML toolbox [33]. However, this too allows limited control over the type of data being exchanged, the choice of backend learning framework and the version of the learning framework.

Deepmind in collaborate with Unity3D have also developed their own toolbox for machine learning [34], which forces users to use their cloud based services, over their own local machines, thus making it very inconvenient. This also limits the users to using only Deepmind's learning framework.

Thus, an open source communication library has been developed and designed to work with the Unity Engine. This library allows the user to simply add a script to their agent in the environment. This script is passed the input senses and other environmental variables (such as score and rewards), and it transmits the data over

sockets setup using the TCP/IP protocol. The server library is written in C#, whereas the client library is written in multiple languages (C++, Java and Python), and can be ported to more other languages in the future if required.

The benefits of this library over others are:

1. Freedom to design any environment using the Unity Engine.
2. Freedom to use any backend language for algorithmic development of AI and machine learning tools.
3. Freedom to choose your own learning framework/library/tool.
4. Compatible with any version of libraries and tools in other languages, since TCP/IP is a fairly stable protocol.
5. Ease of use
6. Ability to setup multi-station learning environments. This is because, the sockets allow communication over a network, thus the simulation station and the machine learning station could either be the same or different. They could be on the same network, or on different networks (as long as they have access to the internet). Furthermore, multiple agents could source their backend tools from different stations, as each agent is allocated a unique independent socket for communication.

Chapter 6

Results

Previous chapters have covered the various test environments and the algorithms developed in this thesis. This chapter will state and discuss the results of the algorithms being evaluated on the newly constructed simulation environments. The algorithms are plugged into various agents which are then trained to perform tasks in the environments. Each environment set consists of 4 scenarios/sub-environments/objectives.

The performance of the Relational Neurogenesis (RN) algorithm will be compared to other state-of-the-art and competing algorithms. In this work, we compare this implementation against 5 other algorithms, namely PPO [8], NEAT [7], EWC [4], CHILD [10] and PathNET [35]. The Proximal Policy Optimization (PPO) [8] algorithm can be considered as the primary competitor for evaluating RN as a reinforcement learning algorithm, as it is the current state-of-the-art for the same. The Neuroevolution of Augmenting Topologies (NEAT) algorithm is the most prevalent evolutionary algorithm and is used as a base for this algorithm, and is thus used as a comparison metric for baseline evolutionary mechanisms. The Elastic Weight Consolidation (EWC) [4], Continual, Hierarchical, Incremental Learning and Development (CHILD) [10] and PathNet algorithms are comparison metrics for reinforcement based lifelong learning algorithms. The EWC algorithm also holds the current state-of-the-art for continual learning in Atari Games. A deeper discussion and analysis of these competing algorithms is provided in the previous research section of the background

(ch 2).

As the aforementioned algorithms excel in various different aspects of reinforcement learning, the results section for each environment set will be divided into 3 main categories:

1. Independent Learning:

Evaluation of the algorithm's ability to learn a single task/sub-environment.

2. Continual Learning:

Evaluation of the algorithm's ability to learn multiple tasks/sub-environments sequentially and then recollect them without retraining.

3. Combined Learning:

Evaluation of the algorithm's ability to adapt and survive in a new environment formed by combining two or more previous environments after being trained on the previous environments individually.

All 6 algorithms will compete during independent learning, as all of them are qualified for single-task learning problems. However, PPO and NEAT will not participate in the continual and combined learning tests as they are not designed to handle multi-task learning. Each algorithm is given the same limited hardware resources and a fixed time period during which it is trained on as many episodes as it can accommodate (in the time slot).

6.1 Independent Learning

This section will evaluate the algorithms ability to learn a sub-environment independently within each environment set. An episode is an attempt to learn the environment, which either ends in the agents death, defeat or timeout. The agent receives a score for its performance during an episode. The scores are continuously plotted as

Figure 6.1: Training Environment - A1 Independently.

a function of the episode number during training. The results of the test phase are obtained by averaging the score over 5 runs of a single episode of the fully trained agent within a specific sub-environment.

6.1.1 Independent Environment Set A

While training the algorithm on environment A1 independently, RN is observed to converge to a solution twice as fast as PPO and PathNet, and 50% faster than EWC, and a whopping 150% faster than NEAT and CHILD. It is seen to match EWC in accuracy and nearly attain state of the art with PPO. This can be seen in figure 6.1.

Training the network on environment A2, as seen in figure 6.2, RN attains state of the art performance out performing even the single task algorithms. It can also be noted that RN establishes a linear rate of information growth, showing the stability of the algorithm. However, the sudden drops can be attributed to the baseline NEAT exploratory mechanisms, which sometimes sacrifices stability for exploration.

Training the network on environment A3, as seen in figure 6.3, EWC and PPO perform better than RN, but take note of how RN is able to outpace the group for

Figure 6.2: Training Environment - A2 Independently.

most of the training run, with substantially higher performance at every episode step.

In environment A4, as seen in figure 6.4, RN is noticeably quick to converge, although unable to match PPO, is still able to keep within 1-2% of the performance range. And is able to outdo EWC. CHILD and NEAT are invariably outdated and prove insufficient on their own.

After training, the final performance scores for each algorithm on each sub-environment are visualized here in figure 6.5. We ignore PPO and NEAT metrics, as this comparison is to form a reference for the continual learning results.

6.1.2 Independent Environment Set B

While training the algorithm on environment B1 independently, RN is observed to converge to a solution twice as fast as NEAT and CHILD, and 50% faster than EWC. It is seen to match EWC in accuracy as can be seen in figure 6.6. The linear rate of growth is observed in environment B1. This reflects the nature of the task which is essentially learning to navigate through the forest and avoid fires. This is a fairly linear task as the relationship between the fire and temperature is linearly

Figure 6.3: Training Environment - A3 Independently.

Figure 6.4: Training Environment - A4 Independently.

Figure 6.5: Testing results on Environment Set A.

Figure 6.6: Training Environment - B1 Independently.

proportional to the reward.

Training the network on environment B2, as seen in figure 6.7, leads to significantly higher accuracy, as the task builds on environment B1. We can see that the virtual integration with the simulated FLIR camera and the audio processor are a success as the algorithms are able to locate the stranded people. RN is seen to have a sudden accuracy bump near the 1200 epoch. It is observed that at this point it learns that it is more rewarding to move away from previously located people towards finding other trapped people. Thus the reward bump reflects its decision to start rescuing more people.

The training of the network on environment B3, is seen in figure 6.8. RN achieves state-of-the-art performance on this task beating both single and continual learning algorithms in measures of accuracy and learning rate. It is observed that all algorithms seem to report sudden large increments in score between the 900-1200th episodes. This happens as a result of the algorithm controlled drone finally rescuing a person and receiving a quadrupled reward for the same. This is very noticeable in the RN algorithm, as it leaps from an average score of 15 to 50 at episode 981. As

Figure 6.7: Training Environment - B2 Independently.

rewards are sparse in this sub-environment, the average overall scores are also lower as compared to other sub-environments.

In environment B4, as seen in figure 6.9, RN is seen to match other algorithms in terms of accuracy. A similar sudden performance bump is seen in the center as environment B4 is an extension to B3. However, the rewards for rescue are lowered in this environment. Therefore this increment can be attributed to the fact that the drones begin working together. However, this is not very well observed in the environment. The drones do seem to cooperate with one another, but collective behavior is not often observed. Additional incentives or behavior modelling might be needed to elicit this response from the system, or the algorithm might simply require longer to stumble upon this advantage by itself.

After training, the final performance scores for each algorithm on each sub-environment are visualized here in figure 6.10. We ignore PPO and NEAT metrics, as this comparison is to form a reference for the continual learning result.

Figure 6.8: Training Environment - B3 Independently.

Figure 6.9: Training Environment - B4 Independently.

Figure 6.10: Testing results on Environment Set B.

6.2 Continual Learning

This section evaluates the algorithms ability to recollect learned knowledge of previously learned tasks. In the context of lifelong learning, it is essential to measure the networks ability to learn tasks in a sequential manner and then be asked to repeat the same tasks in the same or jumbled order without any retraining on said task. This also measures the forgetfulness of a network (catastrophic forgetting).

In each subsequent subsection 6.2.1 6.2.2, the algorithms will be trained sequentially on sub-environments E1, E2, E3 and E4 in that order. Where E1 (environment 1) refers to the first environment in the set. Thus $E1 = A1$ for environment set A. Only the results of testing are seen in this section, where testing is carried out on the same environments in the same order as training. Here neurogenesis and learning sections of the algorithms are disabled, and hence the order of testing is independent of the results.

6.2.1 Continual Environment Set A

While evaluating Environment Set A for Lifelong Learning capabilities, RN attains state of the art performance outperforming other algorithms in most sub-environments. It can also be noted that when compared to the independent learning results, RN maintains a less than 10% degradation in performance. Although PathNET performs

Figure 6.11: Testing Environment Set A - Continually.

Figure 6.12: Testing Environment Set A - Continually. [Before and after limiting the network size of the EWC network]

excellently in single task workloads, it shows considerable degradation as more tasks are piled on, losing nearly 5% more on every previous task when a new task is added. This is visualized in figure 6.11.

Initial tests were conducted by giving EWC an arbitrarily large network size (as it does not grow or resize automatically), here EWC is seen to match RN in multiple environments. Note that the EWC network size (400 nodes) here is 33% larger than the maximum RN network size (300 nodes). After curbing the initial size of the EWC network to the maximum size of the RN network (300 nodes), we observe the results in figure 6.12. Here the RN algorithm is seen to gain the upper hand in all sub-environments and tasks.

Figure 6.13: Network Size [no of nodes] of algorithms trained on Environment Set A.

Comparing the network sizes of the networks produced by different algorithms allows us to view the operational efficiency of the neurogenesis (growth) part of the algorithms. Here it is optimal to balance performance and network size. RN maintains significantly lower network sizes in all sub-environments, however PathNET obtains the lowest numbers. Although RN is not the lowest, it best balances the trade-off between performance and size, as PathNET prunes the network down at the cost of performance.

6.2.2 Continual Environment Set B

While evaluating Environment Set B for Lifelong Learning capabilities, RN obtains the highest scoring performance for all sub-environments, except B4. PathNET is seen to nearly match RNs performance, with minimal performance degradation. This is a contrast to environment set A, where PathNET showed severe degradation in previously learned tasks. This is visualized in figure 6.14.

After curbing the initial size of the EWC network to the maximum size of the RN network, we observed significantly better results for environment set A. Thus the EWC network size was initially set to 500 nodes to approximately match the maximum network size of the RN network (495 nodes). Figure 6.15 shows the results of continual learning on the performance. RN manages to maintain low network size

Figure 6.14: Testing Environment Set B - Continually.

Figure 6.15: Network Size [no of nodes] of algorithms trained on Environment Set B.

due to the inherent mechanisms which limit network growth and prune unnecessary nodes. However, PathNET is still able to maintain the best size to performance ratio for environment set B. EWC on the other hand is not able to match the performance of either RN or PathNET despite having the largest network size right from the initialization stage.

Despite PathNET having the lowest "final" network sizes, it should be noted that during training, PathNET uses much larger memory resources to train, as it starts with a large number of nodes and prunes itself down towards a smaller number of nodes. Whereas RN only grows to the required size when needed and uses minimal pruning. This is important because for an online learning approach (which is ideally suited for continual learning algorithms) as the systems memory resources would need to be much larger while training, but then would be wasted and idle after training is completed as the network size only shrinks.

6.3 Combined Learning

This section pertains only to environment set A, which enables us to measure the algorithm’s ability to generalize to a new environment formed by fusing components/tasks from previous environments. This evaluation serves to prevent the algorithm from encoding tasks as separate entities, and rather encourages sharing information and network resources across sub-environments/tasks. Environment set B inherently performs combined learning in environments B3 and B4, and are thus not evaluated separately.

Chapter 7

Conclusion

Thus we reach several conclusions about the various aspects of the research presented here.

First, a quick summary of the contributions and achievements of this work, highlighting the key focus points of the research so far. Followed by further exploration of the various applications to which the work presented here might be applicable. And a note about the optimization of the network for portable deployment. And finally, the mechanisms and modifications which have not yet been implemented, but have been envisioned, planned and are on the horizon.

7.1 Summary

The previous chapters have, in detail, covered all the proposed mechanisms for enabling lifelong learning agents to learn in reinforcement learning environments, starting from an initial review of the state of lifelong learning agents within the domain of reinforcement learning, and a comparison and analysis of several previous approaches.

This thesis addresses several hurdles faced by evolutionary machine learning algorithms tackling lifelong learning; namely catastrophic forgetting, network capacity and evolving network topology. The proposed algorithms encapsulate mechanisms to grow a hierarchical network architecture using reinforcement learning. Various rules are specified for Neurogenesis and Synaptogenesis, and their reverse counter-

parts. These rules allow evolution of network architectures while preserving previously learned information and maintaining network capacity.

The effectiveness of the methodology is compared with state of reinforcement-based lifelong learning mechanisms, and is shown to match their performance at lower episodic cost. It is also shown to beat the state-of-the-art accuracy achieved by the EWC algorithm at the same degree of network density.

However, the overhead complexity in rule determination and computation result in the algorithm requiring larger amounts of resources as compared to competing systems. This results in slower performance. But the efficiency of the algorithm compensates for this by converging proportionally faster, and thus requiring the same time to converge on hardware with limited resources.

7.2 Applications and Portability

As demonstrated through the creation of various virtual environments and the deployment of the algorithm in training agents within them, it is evident that through similar processes, one can model any situation or task to be synthetically tested. With the rise in simulation environment capabilities, the complexity of tasks can be increased to match pseudo-realistic standards.

The goal is to develop these environments and algorithms side-by-side and train them within the virtual setting. The trained model can then be deployed onto a real world agent/robot, which would then be deployed in real-world scenarios. Thus the algorithm is not limited to simulations only. This kind of deployment would require the algorithm to be computationally light and memory efficient. While this algorithm is memory efficient, and minimizes the network growth to smaller models during neurogenesis, it is not comparatively computationally light.

Further research is required to reduce the complex and intensive calculations performed during the execution of the algorithm to make it more suitable for field de-

ployment. However, in its current state, it is suitable for transfer learning, as none of the processing overheads occur during inference. But this strips away the dynamic ability of the network to grow and adapt while in the field.

The algorithm itself has proven to be capable of learning multiple conflicting scenarios and multi-dimensional tasks. It is able to hold its own against other state-of-the-art algorithms and learns quicker than most algorithms per cycle. It has been demonstrated to be suitable for training robotic agents such as drones and spiders (extended to spider-bots), and thus one can extrapolate all the possible robotic applications which it can be trained to accommodate. Ranging from autonomous navigation to search and rescue operations.

Although it can easily be adapted to industrial tasks, such as assembly and maneuvering, the dynamic growing nature of the algorithm (tailored for lifelong learning) is better expressed in the field. The ability to learn continually allows for applications of unknown territory exploration and evacuation. Also, the ability to process input data from multiple sensors shows the range of capabilities to handle various inputs simultaneously in tight situations.

7.3 Future Work

Immediate goals for this work are to reduce the computational needs for the algorithm by trimming down unnecessary operations. However, this will only lead to slight improvement. In order to have a greater impact on the algorithm, future work will have 3 major goals:

Initially, the algorithm should be ported to a spiking neural network model from its current non-spiking form. This would greatly make it more power efficient on suitable hardware, and also enable it to be deployed on more recent neuromorphic platforms [36].

Secondary efforts would be in converting the multifaceted rules into local learning

rules. This would enable more local computation from distributed units rather than more central overhead processes from a single controlling unit. Currently, the entire network is managed by a single control unit and every neuron/node is under its influence. This limits the hardware to a single control unit, and otherwise bottlenecks other internal parallel processes. However, if groups of nodes could be automatically assigned to multiple distributed and independent control units, that would greatly improve the parallelism.

The final goal would be to reduce the number of learning rules to hopefully a single learning mechanism. There is still much debate as to whether a single learning mechanism controls the learning process in biological neural tissue, but there is no proof to support the contrary, and thus the possibility of discovering or inventing such a rule remains open. A single learning mechanism would make the training very efficient on both GPUs and distributed spiking platforms.

Simultaneously, there is a need to develop more comprehensive testing environments to verify the performance of the developed algorithms. The work on the UnityAI communication library will continue to grow alongside the Unity Engine itself. The next iteration of the library will include improved CNN (image frame) support.

Supporting projects have lead to the development of a physical mobile robotic system as seen in figure 7.1. This robot was modelled as a 3D agent alongside a virtual environment of our very own Nu.AI lab at RIT as seen in figure 7.2. The goal being to train the robot to perform various navigation and load bearing tasks within the lab. The robot will be trained in the virtual environment, and then tested in the real lab to verify its ability to transfer learn. This would serve as a test case towards more meaningful endeavours.

Figure 7.1: Nu.AI lab robot in real life (left) vs simulation (right).

Figure 7.2: Simulated environment of the Nu.AI Lab at RIT.

Bibliography

- [1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, p. 354, 2017.
- [3] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller *et al.*, “Emergence of locomotion behaviours in rich environments,” *arXiv preprint arXiv:1707.02286*, 2017.
- [4] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, p. 201611835, 2017.
- [5] L. Wiskott, M. J. Rasch, and G. Kempermann, “A functional hypothesis for adult hippocampal neurogenesis: avoidance of catastrophic interference in the dentate gyrus,” *Hippocampus*, vol. 16, no. 3, pp. 329–343, 2006.
- [6] L. B. Soros and K. O. Stanley, “Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria,” in *H. Sayama, J. Rieffel, S. Risi, R. Doursat, & H. Lipson (Eds.), Artificial life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. Citeseer, 2014, pp. 793–800.
- [7] K. O. Stanley and R. Miikkulainen, “Efficient evolution of neural network topologies,” in *Evolutionary Computation, 2002. CEC’02. Proceedings of the 2002 Congress on*, vol. 2. IEEE, 2002, pp. 1757–1762.
- [8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [9] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart *et al.*, “Relational deep reinforcement learning,” *arXiv preprint arXiv:1806.01830*, 2018.
- [10] M. B. Ring, “Child: A first step towards continual learning,” *Machine Learning*, vol. 28, no. 1, pp. 77–104, 1997.
- [11] —, “Continual learning in reinforcement environments,” Ph.D. dissertation, University of Texas at Austin Austin, Texas 78712, 1994.

- [12] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search,” *arXiv preprint arXiv:1806.09055*, 2018.
- [13] J. K. Pugh and K. O. Stanley, “Evolving multimodal controllers with hyperneat,” in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 735–742.
- [14] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, 1989.
- [15] S. Džeroski, L. De Raedt, and K. Driessens, “Relational reinforcement learning,” *Machine learning*, vol. 43, no. 1-2, pp. 7–52, 2001.
- [16] T. Croonenborghs, K. Tuyls, J. Ramon, and M. Bruynooghe, “Multi-agent relational reinforcement learning,” in *Learning and Adaption in Multi-Agent Systems*. Springer, 2006, pp. 192–206.
- [17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.
- [18] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci, “A hypercube-based encoding for evolving large-scale neural networks,” *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.
- [19] S. Risi, J. Lehman, and K. O. Stanley, “Evolving the placement and density of neurons in the hyperneat substrate,” in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 563–570.
- [20] S. Risi and K. O. Stanley, “A unified approach to evolving plasticity and neural geometry,” in *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, 2012, pp. 1–8.
- [21] M. Ring, “Learning sequential tasks by incrementally adding higher orders,” in *Advances in neural information processing systems*, 1993, pp. 115–122.
- [22] —, “Sequence learning with incremental higher-order neural networks,” *University of Texas at Austin, Austin, TX*, 1993.
- [23] F. Zenke, B. Poole, and S. Ganguli, “Continual learning through synaptic intelligence,” *arXiv preprint arXiv:1703.04200*, 2017.
- [24] X. He and H. Jaeger, “Overcoming catastrophic interference using conceptor-aided backpropagation,” 2018.
- [25] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, “Progress & compress: A scalable framework for continual learning,” *arXiv preprint arXiv:1805.06370*, 2018.

- [26] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, “Real-time neuroevolution in the nero video game,” *IEEE transactions on evolutionary computation*, vol. 9, no. 6, pp. 653–668, 2005.
- [27] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [28] Z. Li and D. Hoiem, “Learning without forgetting,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.
- [29] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, “Progressive neural networks,” *arXiv preprint arXiv:1606.04671*, 2016.
- [30] D. Lopez-Paz *et al.*, “Gradient episodic memory for continual learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6467–6476.
- [31] Unity Engine, “Unity3d.” [Online]. Available: <https://unity3d.com/>
- [32] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [33] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A general platform for intelligent agents,” *arXiv preprint arXiv:1809.02627*, 2018.
- [34] Unity3D and Deepmind, “Deepmind unity toolbox.” [Online]. Available: <https://blogs.unity3d.com/2018/09/26/unity-and-deepmind-partner-to-advance-ai-research/>
- [35] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, “Pathnet: Evolution channels gradient descent in super neural networks,” *arXiv preprint arXiv:1701.08734*, 2017.
- [36] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, “Loihi: a neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

Glossary

agent

Any entity that can interact with an environment. It receives inputs from environmental phenomena and interacts by means of its actuators/outputs as a direct or indirect consequence of those inputs.

drone

An unmanned/remotely-controlled vehicle or robotic agent.

edge

A weighted connection between two nodes. Analogous to a synapse in biology.

framework

A generic supporting software tool/system/library which can be used for development. A user might be able to modify its functionality and operation through custom code.

game engine

A software framework which is dedicated to running video games by explicitly supporting visualization graphics, real-time physics calculations and other gaming peripherals.

layer

A set of nodes which constitute a section of a network's internal hierarchy.

neuroevolution

Process of a neurogenesis and synaptogenesis in a network.

neurogenesis

Generation of additional neurons/nodes.

node

Computational region of convergence, where edges connect to and originate from, and where an activation function is applied. Analogous to a neuron in biology.

reinforcement

Process of consolidating concepts/information by repeated exposure to relevant data.

simulation

The recreation of a real world process in a digital environment.

spawning

The process of introducing an object into a simulation running on a game engine at or during run-time.

synaptogenesis

Generation of additional synapses/edges/connections.

TCP/IP

Transmission control protocol/Internet protocol, used to govern the connection of computer systems to the Internet.