Evolving developmental, recurrent and convolutional neural networks for deliberate motion planning in sparse reward tasks

Benjamin Paul Jolley

A thesis presented for the degree of
Doctor of Philosophy

Submitted for the degree of
Doctor of Philosophy
June 2020
Keele University

Acknowledgements

I would first like to express my appreciation to the numerous people that have been supportive throughout this journey. The list would be bountiful but know that any help, advice, discussion or simply being present during this period was fully appreciated. My PhD supervisor, Dr. Alastair Channon, has provided a wealth of knowledge throughout whether that be academic or personal. His guidance created a riveting and enjoyable exploration of the research field and his optimistic perspective of my endeavours allowed me to get through moments of dismay when they inevitably arose. I have been incredibly fortunate to have him. My gratitude to Dr. James Borg who steered me on this path as my undergraduate dissertation supervisor. It was those early seeds which led me here and I'm sure without his recommendation this would not have been a possibility. James also had the misfortune of being a co-author on my first paper during my academic writing infancy; I commend his patience. Shout out to Ben Jackson for the shared PhD experience, we got to simultaneously participate in the daunting, but brilliant, time this can be. The whole artificial life community provides an incredibly interesting and fulfilling research area, it has been a pleasure to be a part of and meeting the people within it. I would like to thank my examiners, Prof. Peter Andras and Dr. Julian Miller, who provided fair and constructive comments and criticisms which resulted in an overall stronger body of work. I would also like to thank the staff and students I have encountered at Keele University, it has been gratifying to be around such insightful people. Finally, my parents are owed the utmost praise for everything was not possible without them. They have provided nothing but love and support and I greatly appreciate them.

Abstract

Motion planning algorithms have seen a diverse set of approaches in a variety of disciplines. In the domain of artificial evolutionary systems, motion planning has been included in models to achieve sophisticated deliberate behaviours. These algorithms rely on fixed rules or little evolutionary influence which compels behaviours to conform within those specific policies, rather than allowing the model to establish its own specialised behaviour. In order to further these models, the constraints imposed by planning algorithms must be removed to grant greater evolutionary control over behaviours. That is the focus of this thesis.

An examination of prevailing neuroevolution methods led to the use of two distinct approaches, NEAT and HyperNEAT. Both were used to gain an understanding of the components necessary to create neuroevolution planning. The findings accumulated in the formation of a novel convolutional neural network architecture with a recurrent convolution process. The architecture's goal was to iteratively disperse local activations to greater regions of the feature space. Experimentation showed significantly improved robustness over contemporary neuroevolution techniques as well as an efficiency increase over a static rule set. Greater evolutionary responsibility is given to the model with multiple network combinations; all of which continually demonstrated the necessary behaviours. In comparison, these behaviours were shown to be difficult to achieve in a state-of-the-art deep convolutional network.

Finally, the unique use of recurrent convolution is relocated to a larger convolutional architecture on an established benchmarking platform. Performance improvements are seen on a number of domains which illustrates that this recurrent mechanism can be exploited in alternative areas outside of planning. By presenting a viable neuroevolution method for motion planning a potential emerges for further systems to adopt and examine the capability of this work in prospective domains, as well as further avenues of experimentation in convolutional architectures.

Contents

List of Figures

1 | Introduction

1.1 Overview

This thesis examines various neuroevolutionary approaches for motion planning in an artificial neuroevolutionary system to exhibit robust, deliberate, and efficient behaviour. The objective of this thesis is to explore cutting edge neuroevolutionary techniques and contribute to our understanding of current obstacles for evolved motion planning, as well as providing working examples to overcome them. Throughout the thesis, there will be a focus on abstracting away from specialised models that demand significant domain expertise in favour of general-purpose solutions that may adapt across a variety of domains.

The phrase 'motion planning', or 'planning algorithms', encompass terms from robotics [30, 125], control theory [110, 3], computer graphics [25, 120] and more [126]. Thus for clarity, motion planning in this work is framed from an artificial life perspective where simulated animals or autonomous robots convert high-level specifications into low-level descriptions of how to move. The goal in this work is to explore evolvable motion planning techniques to find collision-free paths to multiple desirable points within a varying environment. Work with autonomous robots has provided a great number of literature for motion planning techniques and often is divided into three approaches; probabilistic roadmaps, cell decomposition and artificial potential [125]. Each approach uses the concept of a configuration space where physical space is translated to a finite virtual space. Probabilistic roadmaps create a line segment between every vertex of every object in the configuration space, excluding those which enter the interior of another object [113]. As a result, many unbroken paths should be found to the objective, further algorithms can be used to find a specific type of path. Cell Decomposition subdivides the free space of the configuration space into smaller regions called cells [255]. A connectivity graph is then created which represents the adjacency relations between cells. Through this, an unbroken path can be created from starting position to end goal. This is the basic implementation and does not cover extended works like approximate or quad-tree decomposition. Artificial potential, the technique this particular thesis is interested in, creates potential fields around objects in the environment [117, 247]. Desirable objects produce attractive potential and obstacles generate a repulsive potential. With the combination of these two rules robots aim to approach the goal while avoiding the obstacles. This particular approach has shown its effectiveness in real time navigation due to little pre-computation [148, 30], as well as being a practical application in 2D and 3D artificial life environments [192, 20, 222, 223, 109, 21]. Despite the abilities of each approach, each method's behaviour is dictated by a static set of rules. To become more in line with artificial life concepts we would have to remove this static behaviour in favour of a neuroevolutionary approach. There have been many examples of evolutionary involvement with motion planning but mainly these are constrained to highly guided environments, in which either; the fitness value contains the path's euclidean distance; multiple fitness functions are used for a highly optimal path; the path is already established between object and destination; evolution is used as a parameter optimisation tool with established static models [205, 136, 29, 259, 261, 269, 262, 53, 240, 240]. It is the goal of this work to remove restraints for expanded neuroevolutionary freedom in the creation of viable motion planning solutions. Further, experiments will not be manufactured to guide evolution to a particular behaviour and instead use previously validated environments and tasks. The predominant focus will be on tasks with sparse rewards.

The accepted definition of a sparse reward task is one which gives little to no reward during simulation; as, a precise sequence of events are required in order to receive rewards. As a result, sparse reward tasks are difficult as there are no clear gradients to the desired end goal and therefore greater overall exploration is required. Arguably, the most researched type of problem has been in the Reinforcement learning (RL) field with the Atari video games Montezuma's Revenge and Pitfall and is still a contemporary problem to solve [12, 171, 230, 84, 143, 173, 24, 193, 176, 224, 57]. Sparse reward domains are ideal for seeking general-purpose solutions as there should be no inherent bias to steer evolution towards an engineered desired result. This will then focus the research towards intrinsic behaviours in the neuroevolutionary process, whether that be in the architecture, population or both.

As mentioned previous, the work intends to frame itself from an artificial life perspective. A typical intended goal for artificial life systems is the use of tools to study the foundations of life and evolution in a setting other than natural biology [124]. Possibly the most infamous quote associated with artificial life is by one of the founders, Chris Langton, who encompasses this vision in the following excerpt.

"*Artificial Life is the study of man-made systems that exhibit behaviors characteristic of natural living systems. It complements the traditional biological sciences concerned with the analysis of living organisms by attempting to synthesize life-like behaviors within computers and other artificial media. By extending the empirical foundation upon which biology is based beyond the carbon-chain life that has evolved on Earth, Artificial Life can contribute to theoretical biology by locating life-as-we-know-it within the larger picture of life-as-it-could-be.*" [123]

Focusing the research area specifically on motion planning may appear as an engineering challenge as opposed to a natural biological phenomena to observe. Yet, static motion planning is a constraint on any complex system that utilises it. Constraints make certain behaviours achievable but they limit the creativity and ability to generate novel behaviours [134]. Achieving more sophisticated behaviours, and models, requires the removal of constraints. Take for example different types of animal locomotion (i.e. terrestrial, aquatic, aerial, etc) which can then be broken down into greater sub-categories (i.e. bipedalism, quadrupedalism, jet propulsion, anguilliform, gliding, powered flight). Each would require its own motion planning strategy to account for their unique locomotion interaction with the environment.

1.2 Structure of Thesis

- Chapter 2 contains a literature review of neuroevolution, non-neural and non-evolutionary techniques relevant to this thesis. Three subsections are presented; direct encoding, indirect encoding and maintaining diversity. Each section builds a historical narrative from early to state-of-the-art techniques. Each technique will receive; an explanation, example application/s and commentary on successes or shortcomings. A discussion is provided in the final section. This expresses the author's views for the function of these techniques for the thesis goal.

- Chapter 3 presents the two domains and networks that will be utilised throughout the rest of the thesis. Firstly, a hierarchical domain where high-level deliberative and reactive behaviours are produced by a modular neural architecture. This section introduces the static motion planning network which future chapters aim to replace with an evolvable solution. Secondly, a common benchmarking platform, with a variety of domain types, for evaluating the general-purpose competence of an agent.

- Chapter 4 presents the first contributions to the research. Two methodologies are taken from chapter 2. Each is evaluated on their motion planning abilities within the domain introduced in chapter 3. This chapter first examines the complication of scalable motion planning tasks and focuses on agent's ability to scale. The domain is scaled to the smallest possible size before scaling up to the original size. Then, the second experimentation examines the qualitative results of the produced solutions; including training, robustness, and efficiency. The work in this chapter has been presented at IEEE Symposium Series on Computational Intelligence 2017 (Honolulu, Hawaii) and published in Jolley and Channon [107].

- Chapter 5 demonstrates a novel network design to overcome the shortcomings in previous chapters. For this, a convolutional neural network is used with a recurrent convolution process. This method showed a significant improvement in robustness over previous techniques. The efficiency of paths rivals that of a static approach. The implication of this chapter is that a novel use of the existing convolutional neural network architecture can produce robust and efficient motion planning that is unattainable by current known neuroevolution methods. The work in this chapter has been presented at The Conference on Artificial Life 2018 (Tokyo, Japan) and published in Jolley and Channon [108].

- Chapter 6 extends the validity of the findings in Chapter 5. The same recurrent convolutional neural network is evolved in combination with other networks. The aim is to move from a strict planning problem to a domain which requires multiple behaviours; one of which is motion planning. The end result would be an evolvable model that can show long term deliberate behaviours in a sparse reward task. The additional networks are stripped of domain-specific behaviour to contribute to the difficulty. The findings show that high levels of robustness and efficiency, in completing the task domain, are still achievable, despite the increased evolutionary difficulty. This achievement is further highlighted with a comparison with a current state-of-the-art general game player; it also attempts the task domain, with no success.

- Chapter 7 expands the recurrent convolution process to conventional convolutional architectures. The aim is to observe whether recurrency provides benefits to other domains. For this, a common benchmarking platform is used which provides 12 diverse environments. The results demonstrate that recurrency is not restricted to small scale networks. Improvements were seen on multiple domains in the benchmarks. As well as, outperforming other learning techniques and architectures.

- Finally, Chapter 8 provides an overview of the thesis. Each chapter is examined and the implications of the findings are discussed and related to the broader research field. Future work is also considered.

2| Neuroevolution

In the pursuit of artificial general intelligence, researchers have taken inspiration from the natural world and adapted them in silco. Through this process, core components have been established as being functional abstractions such as neural networks; computational structures modelled from rough abstractions of animal brains [95]. Beyond this, research fields work coincide using different paradigms to achieve the overarching goal of intelligence. One particular strategy to create intelligent models is to utilise a learning process that is human-engineered. This requires human expertise and is specific to the task; learning methods such as supervised learning and model-based reinforcement learning. Supervised learning requires human insight to appropriately label input data to achieve a corresponding output. This method of learning is conventionally paired with backpropagation [250]; which calculates the loss function gradient and when used in combination with Stochastic Gradient Descent (SGD) the network weights are modified to reduce the loss. Model-based reinforcement is able to take a model's state and action to predict the next state and the next reward. This is achieved through a predefined function which can be leveraged before learning to produce an effective advantage (i.e knowing the mechanics of a domain prior). As a result, the model can consider possible future situations before they are actually experienced. Another approach is to utilise algorithms that purely learn from trial-and-error experience. Methods such as; model-free reinforcement learning to either optimise an action-value function [156] or policy directly [158]; Evolutionary Strategies uses a population of policy parameters, created from an initial parameter set, then a standard deviation of noise is applied. The population continually moves to higher expected fitness [254]. These methods can be considered gradient-based as they all calculate or approximate gradients and optimise those parameters via stochastic gradient descent/ascent. Instead, the work in this thesis focuses on a gradient-free optimisation technique inspired by biological evolution, Neuroevolution (NE).

NE trains models with Evolutionary Algorithms (EAs) [159, 266]. EAs mimic the biological process of evolution whereby a population of genomes are bred to be fitter by mutation and recombination of the genetic code [49]. The genome itself could represent many aspects of the evolutionary process, including; network weights, activation functions, network topology, etc. However, the Conventional NeuroEvolution (CNE) approach is a fixed topology and simply evolves the network weights. Due to the low dimensionality of these initial neural networks, NE was prone to premature convergence at local optimums. So, new methodologies were established to encourage greater complexity. CNE is classified as direct encoding, with a clear relationship from genotype to phenotype (ANN). This umbrella term also includes: GAs which support plasticity in the genotype and neuron level optimisation. Indirect and/or developmental encodings allows information in the genome to be reused to affect many parts of the phenotype

[217]; this allows EAs to exploit regularities and scalability. Populations that diversify genetically or behaviourally encourages greater coverage of the search space to discover new promising gradients. The remainder of this chapter will expand the details in this overview.

Finally, a brief section of notable NE contributions from a spectrum of research fields. In robotics, NE has a close relation to evolving controllers for embodied robots [144]. A notable use of a GA was the optimisation of the gait for the Sony AIBO robot [102]. Gaits were simulated with the on board hardware and produced gaits faster than those created by hand. The success of this research saw a commercial release of the robot. Work in Lipson and Pollack demonstrated that with neural control and mechanics simulated concurrently, robots could be 3D-printed to achieve functional movement in the real world [138]. In particle physics, NE produced the most accurate mass estimation models of the top quark [1, 2]. In medical research, GAs have been used in prediction models for; melanoma [206], lung cancer [106], critically ill patients [56]; and computer support diagnosis of skin tumours [87].

2.1 Direct Encoding

Direct encoding involves the genotype having a one-to-one mapping with the phenotype. These NE methods write weights, or weights/topology, to a bit string which then translates to the network architecture. Direct encoding provides the advantage of having a clear understanding of how the network constructs from the genotype representation. Yet, the size of the network has a direct correlation to the size of the genotype representation and causes issues at scale; greater computation time is necessary as the genotypes grow. This could be considered especially a problem with Topology and Weight Evolving Artificial Neural Network (TWEANN) methods, as the topology and connections grow unbounded. This issue became especially clear in the first TWEANN technique to be published, Structured Genetic Algorithm (sGA).

2.1.1 Structured Genetic Algorithm (sGA)

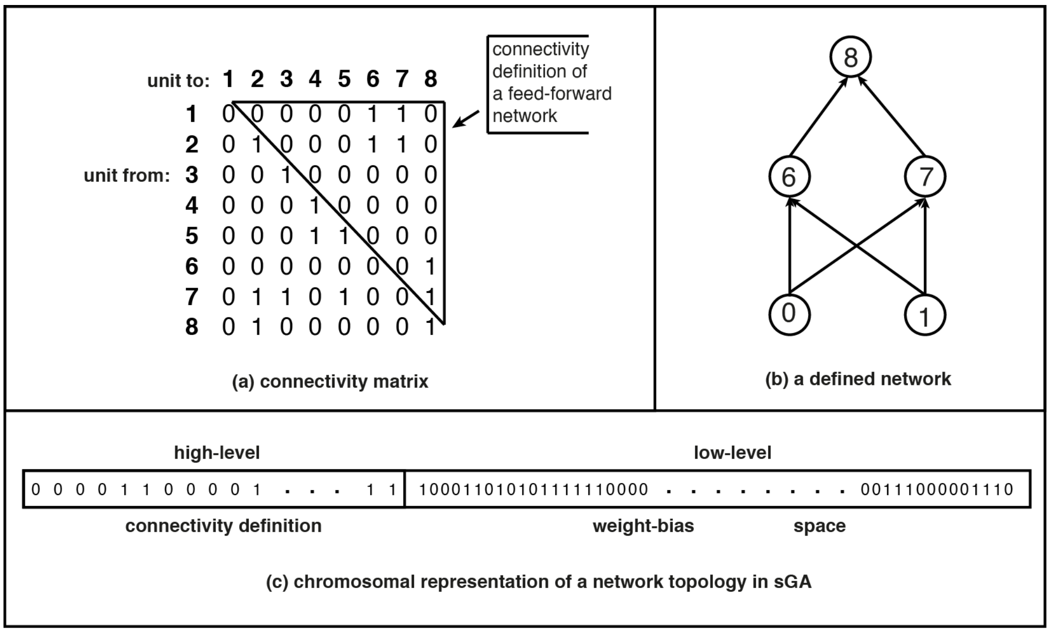


Figure 2.1: **A two-level sGA representing a neural network.** Figure adapted from [45].

sGA combines both structure and weights of the network into a bit string genotype. This approach looks to avoid the 'trial-and-error' processes to find near-optimal network architectures by evolving the topology and weights [47]. To achieve structural mutation, sGA uses a multi-level genetic structure, corresponding to those like a directed graph or tree; the preliminary work focused on two-level structures. The gene levels have a hierarchical relationship to each other, where a number of lower level nodes have a connection to higher level nodes. This is due to the addition of genes becoming active or passive in mutation. High level genes activate or deactivate sets of lower level genes. Therefore, single changes at higher levels can produce multiple changes at lower levels in terms of genes which are active. During recombination, genes stay within the genotype in a redundant form for potential use in future generations. The hierarchical structure allows both long jumps in mutations and precise low-level mutations depending which level mutates.

$$A =< S_1, S_2 > \tag{2.1}$$

A formal mathematical definition of sGA's genome type can be seen at equation 2.1 and is taken from Dasgupta and McGregor [46]; where A represents an ordered set which consists of two strings S_1 and S_2; the length of S_2 is a multiple of the length of S_1 (i.e $|S_1| = s$ and $|S_2| = sq$).

Initial use of sGA showed good results in small scale tasks such as producing an XOR gate [45] but inefficient at scaling due to the connectivity matrix size being the square of the number of nodes. As a result, representation can become huge with greater nodes. Further, the bit string has a fixed length, which limits connection possibilities. For optimal bit string length a trial-and-error process is then necessary and this ultimately is against the purpose of sGA. The next major method at evolving topology and weights was GeNeralized Acquisition of Recurrent Links (GNARL).

2.1.2 GeNeralized Acquisition of Recurrent Links (GNARL)

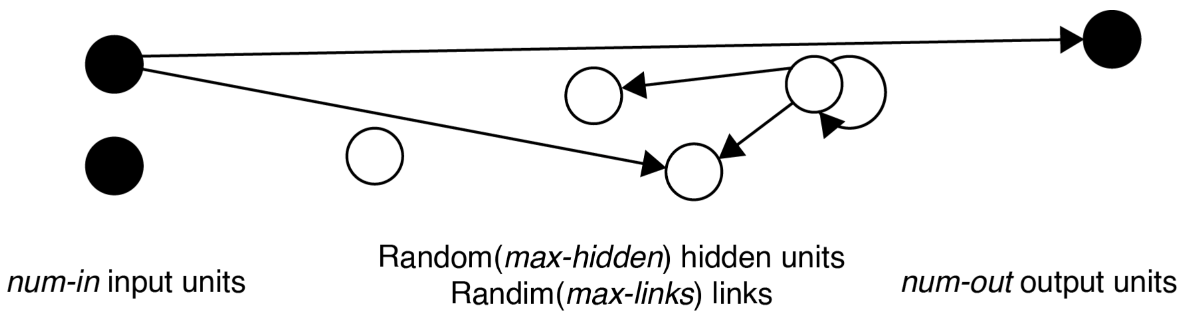


Figure 2.2: **Sample of an initial GNARL network.** Figure adapted from [196].

GNARL is a NE method that non-monotonically constructs recurrent networks to solve task domains [196]. Initially, only the inputs and outputs are present in the network but disconnected. GNARL then defines the connections via the genome. GNARL's genome contains a user defined range of disconnected hidden nodes. Random weights with random values are then added. Connections are restricted to an output node and from an input node. Mutations allow the reuse or removal of nodes and connections; as well as change in connection weights. Removal of neurons also removes the attached connections. New nodes initialise as unconnected and new connections initialise at weight 0.0. GNARL introduces a temperature value which calculates the performance of a network which in turn determines the mutation rate; a concept inspired by simulated annealing [22]. Network's temperature is calculated via the following [4]:

$$T(\eta) = 1 - \frac{f(\eta)}{f_{\text{max}}} \tag{2.2}$$

where f_{max} is the maximum fitness for a given task and η is the given network. High temperature indicates poor performance and mutates the network at greater severity. Low temperatures therefore, indicates good performance and mutates slightly. Criticism of GNARL relates to genetic bloat; in which nodes in the network do not contribute to the overall result. GNARL can produce instances in which neither the input or output layer connect to any other neurons [215].

2.1.3 Symbiotic Adaptive NeuroEvolution (SANE)

SANE evolves feed-forward neural networks with constrained topology evolving elements [162, 160]. As opposed to a genotype of weight matrices, SANE evolves individual neurons. The genotype is comprised of connections and the weights of those connections. SANE employs a symbiotic evolution approach; each member of the population is only a partial solution. In combination with other members, full solutions are achievable. For this to be effective, neurons must develop a symbiotic relationship where they do not diminish the performance of other neurons. Networks construct by selecting a set number of random neurons from the population. Networks are then attempted on a task domain, receiving a fitness value. Each participating neuron in

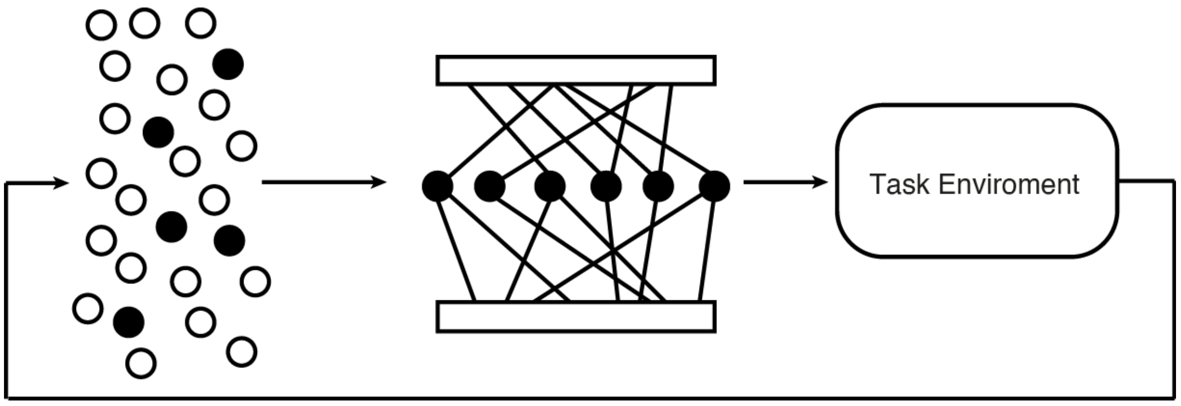


Figure 2.3: **SANE's evolutionary process.** SANE maintains a population of neurons and evaluates each in conjunction with other neurons. Step 1 (the evaluation step) in SANE is broken into three sub-steps. Neurons are continually combined with each other and the resulting networks are evaluated in the task. Each neuron receives a normalised fitness based on the performance the networks in which it participates. Figure adapted from [71].

the network is assigned the fitness score. This process continues until all neurons have been visited a minimum number of times. The neuron's final fitness is calculated by taking an average of fitnesses achieved. Once obtained, a random individual from the top quarter of the population is paired with a random individual with equal or higher average fitness. Mutation is then applied. Mutation rates are set low to introduce genetic material that was not in the initial population or lost during crossover. SANE aims to maintain diversity without expensive operations or high degree of randomness; which high mutation would cause. The offspring replaces the worst-performing neurons in the population. This process repeats until a new population is created.

Problems are prevalent in SANE's initial introduction, as pointed out by the authors [163]. First, fit neurons may crossover with neurons which do not work well together. Thus, a good neuron may be lost due to ineffective crossover. The second, network fitness varies greatly throughout evolution. During early generations, this promotes exploration of the search space. Yet, later generations are unable to focus on optimising the best networks. SANE's inconsistent networks often stall the search and prevent the global optima from being located. These problems only become clear in those task domains which demand high precision within the solution space. So, the original authors extended SANE's functionality.

Hierarchical SANE (HSANE) combines the advantages of network level and unit level evolution. HSANE keeps two populations, one for the units, and another for network blueprints. The network blueprints organise neurons into workable groups. With the additional information, better-performing neurons are assigned a greater number of trials. By keeping a 'memory' of network combinations, explorative search can continue to exploit the best neuron combinations. As opposed to SANE, where desirable combinations are lost. HSANE has been demonstrated on tasks like Go [183] and avoidance in a Robot Arm [161].

SANE and HSANE suffer from being only a part evolving topology method, as both are bound to predetermined parameters. Such as; the number of layers, the number of hidden-nodes and the total number of connections made to the input and output layers. These constraints are not placed upon other TWEANN methods. Although inferior on benchmarks such as pole balancing tasks [216], there are examples of SANE outperforming other NE methods, such as NEAT (section 2.1.5) [204]. But, the extended work for Enforced SubPopulations (ESP), while having the same limitations, provides superior performance [72, 204].

2.1.4 Enforced SubPopulations (ESP)

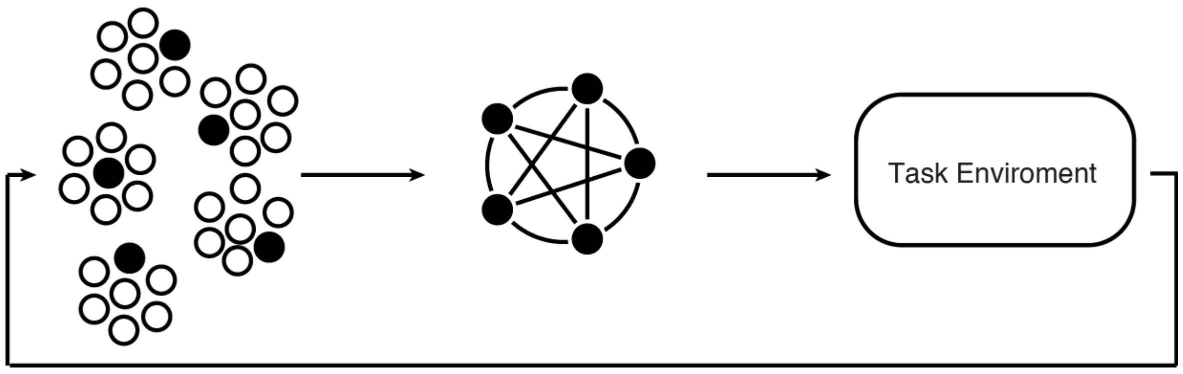


Figure 2.4: **ESP's evolutionary process**. The population of neurons is segregated into sub-populations, shown here as clusters of circles. The network is formed by randomly selection one neuron from each sub-population. Figure adapted from [71].

ESP follows the same methodology of SANE. But, ESP uses speciation to separate each neuron into its own subpopulation [71]. Crossover takes place between neurons of the same subpopulation; the offspring remains in their parents' subpopulation. The full network architecture is constructed by taking a neuron from each subpopulation. This creates an evolutionary pressure for each subpopulation to specialise to a niche. Although the pressure to specialise does apply to SANE at later generations, ESP accelerates the process. ESP's progressive specialisation is not burdened by recombination across one population, in which two individuals can be behaviourally different. As seen in benchmarks, ESP takes substantially less generations to achieve solutions compared to SANE [75, 77, 73]. Next, ESP allows for more effective evolution of recurrent connections. Recurrent connections behaviour is dependent upon the neurons to which it is connected. SANE's recombination of neurons are selected randomly from a single population, so similar neuron combinations are unlikely to be persistent across generations. This also affects HSANE as crossover across one population could still produce a neuron with vastly different behaviour. With subpopulations, neurons will depend on the behaviour of a subpopulation rather than random neurons. This allows recurrence to evolve with the high assumption of receiving a particular behaviour. This has allowed ESP to evolve Long short-term Memory (LSTM) cells [71, 78, 199, 200]. ESP has found success in traditional task domains such as RoboCup soccer [251] and pursuit-evasion games [267]. As well as novel ones, such as active rocket guidance [77]. ESP continually outperforms other direct NE methods on each pole balancing task, including NEAT and SANE [75, 76, 73, 74]. Although, ESP appears to be consistently bested by CoSyNE (section 2.1.7).

2.1.5 NeuroEvolution of Augmenting Topologies (NEAT)

The seminal work for direct encoding TWEANN methods was with the introduction of NEAT [219]. Across the literature, NEAT has been successfully applied to various task domains; robot duel [151], crash warning system [212], the board game Go [218]. The initial population are small, simple networks. Then, over generations those structures complexify. Leading to increasingly sophisticated behaviours, above optimisation. Crossover, speciation, and complexification are the core elements of NEAT. Each were assessed via a knockout tournament on the pole balancing task without velocity. All were shown to have a statistical significance on NEATs performance [215]. NEAT's novel introduction of historical markings makes these elements possible.

Historical Markings & Crossover

Previously, crossover was not utilised in other topology evolving methods as there was a belief in the competing conventions ideology. Competing conventions means having more than one way to express a solution to a weight optimisation problem with a neural network. Genomes representing the same solution may not share the same encoding. Crossover would likely produce damaged offspring as a result [215]. In the introduction of GNARL this issue is highlighted; citing the three forms of deception that would prevent crossover from being effective [4].

1. Two such networks need not have the same bit string representation.
2. Identical topologies but greatly different weights.
3. Parents differ topologically, suggesting the complexity of an appropriate interpretation function will more than rival the complexity of the original learning problem.

These considerations are addressed with historical markings; a unique inheritable identifier. Each structural element in the genome is assigned a historical marking. Genes with the same historical markings can be aligned and compared. Those with the same historical markings can generate valid offspring. This avoids complex topological comparisons. Parents with different topologies can match with common ancestors; i.e genomes that share common structural elements. Offsprings randomly select genes

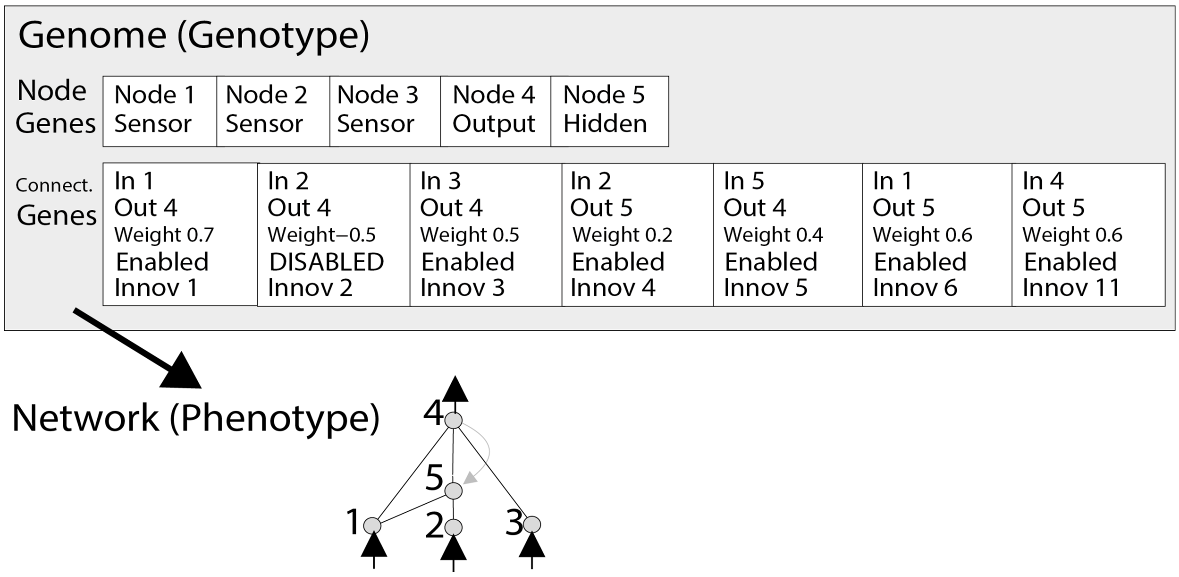

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight−0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3

Figure 2.5: **A genotype to phenotype mapping example in NEAT.** A genotype is depicted that produces the shown phenotype. Notice that the second gene is disabled, so the connection that it specifies (between nodes 2 and 4) is not expressed in the phenotype. Figure duplicated from [219].

from either parent where the historical markings match. Non matching genes (disjoint and excesses) are taken from the fittest parent.

Before NEAT, competing conventions were cited in other literature as an issue. During the knockout challenge, crossover saw less impact than the other aspects of NEAT. Yet, it still produced a statistically significant performance benefit. This suggests that ideas about crossover being detrimental are unwarranted. Crossover is still an avoided process in other TWEANN methods like; EPNET, COVNET, EANT. So, this brings into question why competing conventions are cited as an issue. The two works most attributed to competing conventions are Radcliffe and Xin Yao. Radcliffe claimed a solution for the competing conventions problem would be a *'Holy Grail'* to the field [181]. Xin Yao states "*One of the major advantages of using mutation-based*

EA's is that they can reduce the negative impact of the permutation problem (competing conventions). Hence the evolutionary process can be more efficient." [264]. Yet, both provide no empirical evidence. It seems, even at the time, work had been conducted to debunk these claims. Hancock stating that "*... the permutation problem is not serious in practice*" [86]. Hancock attributes the population size and selection pressures as adequate solutions for a GA to overcome the issue. But seemingly, these initial works swayed research into techniques into avoiding crossover as a result. Yet, a change in structure can initially decrease the fitness of the network. If unaccounted for in the evolutionary process, these potentially beneficial structures are lost to fitter suboptimal structures. Which is why speciation is used.

Speciation

Speciation, also known as niching, is a nature inspired approach to diversify a population and stop convergence on local optimums. Organisms separate into niches in which they compete, as opposed to the entire population. In NEAT, structural innovations are protected within new niches, where they have time to optimise their structure [215]. NEATs implementation of speciation is autonomous. Other uses of speciation have a set amount of species from the offset. This was attributed to the difficulty in comparing topology and weight configurations and how they differed. But, this dilemma is solved with historical markings. The number of excess and disjoint genes between a pair of genomes is a natural measure of their compatibility. The more disjoint two genomes are, the less evolutionary history they share, and thus the less compatible they are. NEAT uses a compatibility distance (δ) calculated via the following [215]:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \bar{W} \tag{2.3}$$

where E represents excess and D represents disjoint genes. $\bar{W}$ is the average weight differences of matching genes. The coefficients (c_1, c_2, c_3) adjust the importance of the three factors, and the factor N, the number of genes in the larger genome, normalises for genome size.

There has been criticism of NEAT's implementation of speciation. The concept of speciation is to functionally diverse a set of solutions for a particular problem. Whereas, diversity in topology allows differing structures while achieving the same behavioural functionality [159]. Alternative uses of speciation have been used previously such as age [100, 201] and fitness [103]. But, a tangible example in NEAT is behaviour-speciation [231]. In which, species are established via a behavioural metric. The results show this allowed more diversity, leading to greater robust solutions in unknown environments. However, the behaviour signature is specific to the problem domain. It requires initial testing in order to find optimal structure for the signatures. Both seem counter-intuitive to the idea of evolving an ANN, which uses evolution as the optimiser.

Briefly touched upon was speciation's relation to bloat, suggesting speciation prevents bloat [215]. Bloat being program growth without (significant) return in fitness [177]; an issue intertwined with genetic programming. An in-depth analysis concluded that NEAT does run bloat free but dependent on the speciation's configuration [232]. So, Bloat-free NEAT (BF-NEAT) was developed to ingrained this into NEAT. In BF-NEAT, species have a higher probability of survival and all species are protected regardless of performance. The result showed an ability to maintain smaller genomes during search, without a substantial decrease in performance.

Complexification

Like the many other concepts in evolutionary computation, complexification is rooted in natural evolution [142]. NEAT not only performs the optimising function of evolution, but also a complexifying function, allowing solutions to become incrementally more complex at the same time as they become more optimal. There are two forms of structural mutation in NEAT which allow complexification. The addition of a node and an addition of a connection. A connection simply connects two nodes which were previously unconnected. New nodes replace the connection between two other nodes with an intermediary node. This is so the introduction of a node affects the network, as opposed to being unconnected. Connections that the node replaces are disabled in the genome. Depending on NEAT's parameters, disabled links have a chance to be re-enabled during crossover. The initial population connects all input to output nodes to avoid genetic bloat. This was a criticism with GNARL's ascertain nodes not being associated with either the input or output nodes. By starting with small, minimal structures and limiting the amount of connections through mutations, NEAT aims to bias the search results to find compact network topologies.

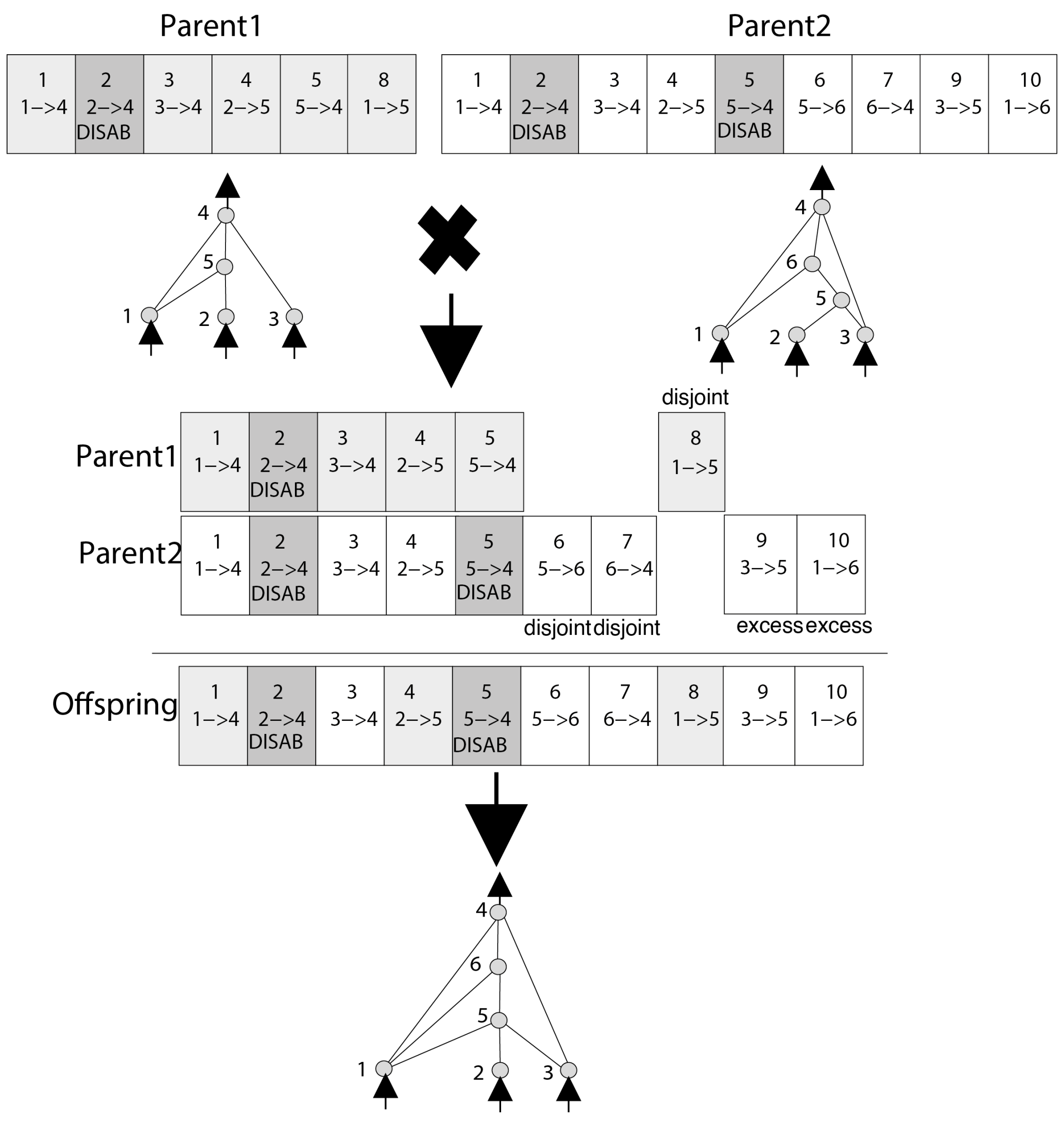


Figure 2.6: **Matching up genomes for different network topologies using innovation numbers.** Although Parent 1 and Parent 2 look different, their innovation numbers (shown at the top of each gene) tell us which genes match up with which. Even without any topological analysis, a new structure that combines the overlapping parts of the two parents as well as their different parts can be created. In this case the parents are equally fit and the genes are inherited from both parents. Otherwise, the offspring inherit only the disjoint and excess genes of the most fit parent. Figure duplicated from [219].

Extensions

Crossover, Speciation, and Complexification have all proved significant to the performance of NEAT and due to their lenient implementation NEAT is able to be modified to suit the task domain. An extension which shows this potential is a real time version of NEAT (rtNEAT) which accounts for user interaction. This particular feature is suited to games in which behaviours can evolve during play, unbeknownst to the player, to adapt to unique play styles. This was demonstrated in Neuro Evolving Robotic Operatives (NERO) [220]. rtNEAT controls the behaviour of the game's opponents. Through the interaction with the environment, opponents learn to navigate mazes, avoid fire and combinations of these approaches. For real-time feasibility rtNEAT changes include; a steady state algorithm, Offspring producing at regular intervals and a fixed number of species. rtNEAT works particularly well in NERO which was credited to an appropriate amount of player interaction with a decent population size of 50. However, some game designs would not meet this prerequisite. Which was shown with Globulation 2, an open-source real time strategy game. With a population size of 7 and limited user interactions, rtNEAT failed to construct efficient controllers within the first 50 generations [172].

A workaround for Globulation 2 was to bootstrap the population with existing, successful controllers. This was also seen in another real time strategy game approach with rtNEAT [64]. The phenotype saves to a database once it exceeds a certain fitness. Bootstrapping the population if evolution doesn't deliver any meaningful progress. But, bootstrapping can bias the search space and leave potentially interesting behaviours unobtainable. Despite these criticisms, if the game design can provide a satisfactory player interaction with opponents rtNEAT is a viable option; in genres such as, first person shooters, fighting games or 2D action games [172]. Games and NE algorithm still have a budding relationship [91, 26, 189, 94] and approaches like rtNEAT could allow enhanced experiences.

2.1.6 NeuroEvolutionary Algorithm (NevA)

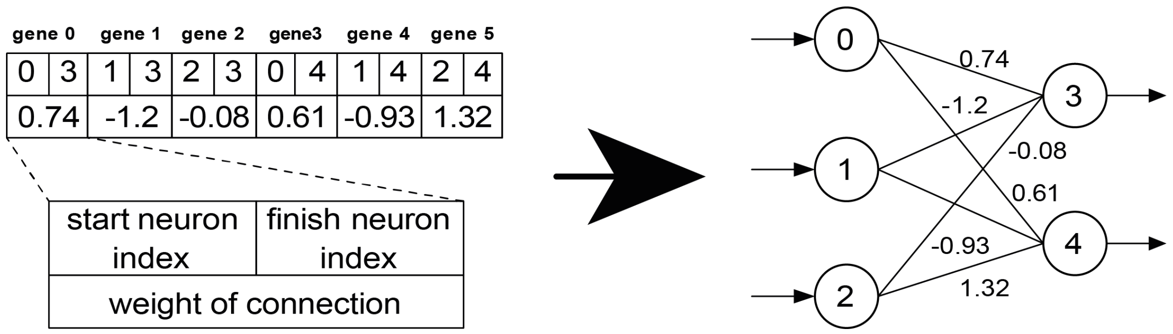


Figure 2.7: **NevA's genetic representation of ANN.** Figure adapted from [233].

NevA directly encodes the connection weights and topology of feed forward and recurrent networks [233]. Each gene in the genome contains the start and finishing neuron and the connection weight. Crossover chooses two parents above average fitness and produces two offspring. Both offspring share common neurons and connections for both parents. The weights of the connections are decided via a 2-point binary crossover. Each connection that differs is randomly chosen to give to an offspring. The initial population of organisms connects all inputs to outputs without hidden nodes. Network weights are initialised randomly in range $[-0.5, 0.5]$. During evolution, the genomes grow structurally to produce more complex behaviour. Mutations can add/remove connections/nodes and set connection weights to random values. Removal or additional connections are either added or removed from the genome. Removal of a neuron requires each instance of the neuron's reference to be removed from the genome. The addition of a connection requires a random starting and random ending neuron. Each generation the fittest is carried to the next without any mutation. Structurally, NevA is similar to NEAT and performs similar on benchmarks. Without speciation, NevA can achieve these results without a large population size. However, NevA has not gained the traction and scrutiny that NEAT has in the literature. So, it is difficult to know if NevA excels past NEAT in other areas.

2.1.7 Cooperative Synapse NeuroEvolution (CoSyNE)

CoSyNE is an encoding scheme in which each network connection has a sub-population. This is in opposition to other cooperative neuroevolutionary methods (e.g. ESP, SANE) which typically use neurons. But, like neuron-level optimisation methods, one member from each sub-population is used in a predefined network topology. So, the number of sub-populations depends on the number of weights and biases. Each sub-population initialises with real numbers, as well as index positions. The given index in each sub-population combine to form the chromosome. Each generation establishes a fitness for all chromosome combinations via a task domain. The population then sorts

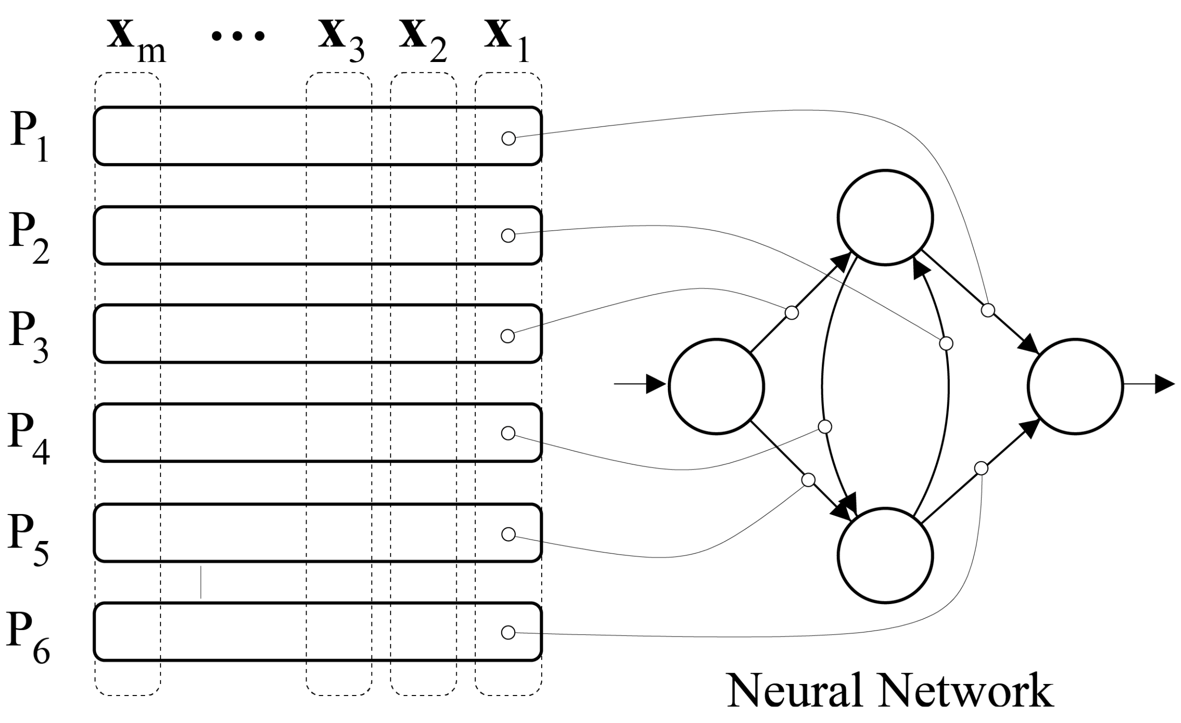

$\mathbf{x}_m$ ••• $\mathbf{x}_3$ $\mathbf{x}_2$ $\mathbf{x}_1$
P_1
P_2
P_3
P_4
P_5
P_6
Neural Network

Figure 2.8: **The CoSyNE method for neuroevolution**. On the left, the figure shows an example population consisting of six subpopulations, each containing m weight values. On the right is the neural network with weights mapped to their corresponding synapses. Figure duplicated from [74].

via fitness. In each sub-population, parents are chosen from the top quarter of fittest individuals. Offspring generate via recombination with crossover and mutation. Recombination produces a pool of new network genes that replace the least fittest in the population. Co-evolving synaptic weights requires rearrangement (permuted) of the sub-populations. So, each weight forms part of a potentially different network in the next generation. Permutation performs probabilistically among their sub-population. How the weights permute are user defined. A sophisticated approach could see disrupting the network proportional to relative fitness. For example, less fit individuals having a higher probability of being permuted. This forces their constituents to search for new complete solutions. This process is expressed in pseudocode below:

```
Algorithm  CoSyNE (n,m, $\Psi$)
  Initialize $\mathcal{P} = \{P_1, \ldots, P_n\}$
  repeat
      for $j = 1$ to $m$ do
          $\mathrm{x}_j \Leftarrow (x_{1j}, \ldots, x_{nj})$
          Evaluate $(\mathrm{x}_j, \Psi)$
      $\mathcal{O} \Leftarrow$ Recombine $(\mathcal{P})$
      for $k = 1$ to $l$ do
          $x_{i,m-k} \Leftarrow o_{ik}$
      for $i = 1$ to $n$ do
          permute $(P_i)$
  until solution is found.
```

In benchmarking domains, CoSyNE demonstrates greater performance than neuron-based sub-populations on the two pole balancing problem without velocity information [74]. However, neuron-based sub-population methods succeed in other domains over CoSyNE, such as pattern recognition [28, 27].

2.2 Indirect & Development Encoding

Indirect and development representations are inspired by biological genes; in particular DNA where a foundation of limited instructions can produce a complex output. [19]. These encodings aim to find the right level of abstraction of biological development to capture its essential properties. Research for these encoding in-silco can be found in early experiments with pattern formation [234, 137]. Developmental and in-direct systems are grounded in the belief that smaller sections make up the larger network. The assumption was that the human brain has to have some modular structure in a similar way that most computer programs use modularity. “*Each procedure is defined a single time and can be called many times*” [81]. The following sections will give an in-depth look at the major contributions to Indirect and development encoding.

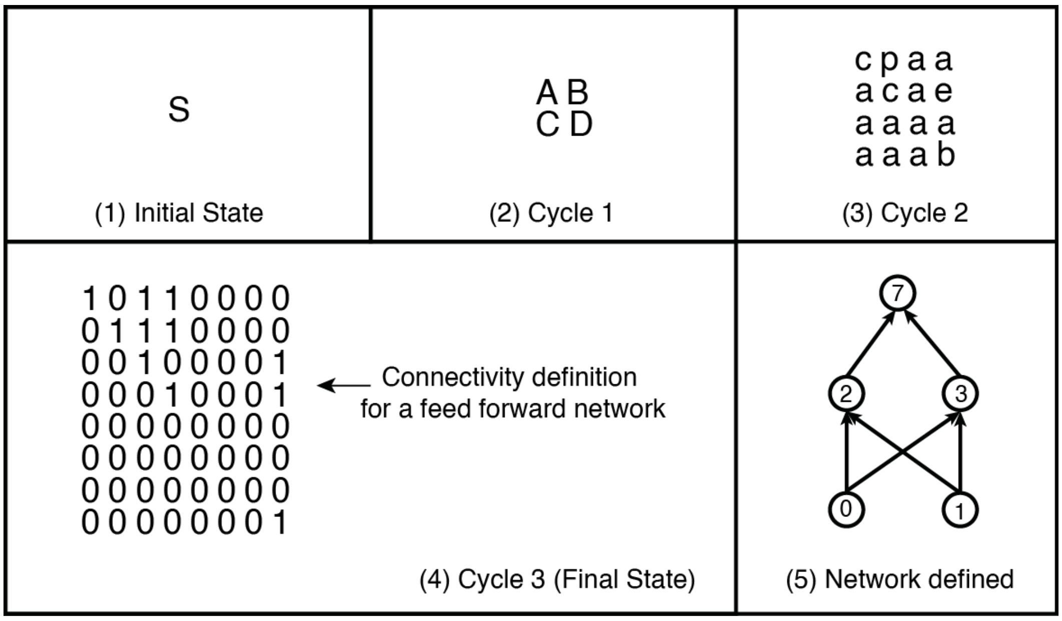


Figure 2.9: **Development of 2-2-1 XOR network with L-systems** . Figure adapted from [35].

2.2.1 Lindenmayer systems (L-system)

L-systems are a parallel string rewriting mechanism [137]. They provide a model and a mathematical theory of plant development; yet, L-systems can replicate many biological processes. Formal grammars are the basis of L-systems. By continuously replacing parts of an initial simple object (axiom) complex objects can form. Productions are the rewrite rules which iteratively rewrite an axiom. Each production rule describes how a certain character, or string, should be rewritten into other characters. L-systems apply all production rules in parallel to form a new string. This is counter to other grammar production rules which apply rules one-by-one. Notable uses of evolution of L-systems are architectural structures [35], robot morphology [101] and modular neural networks [119, 239, 16].

In the creation of neural networks, the GAs population consists of binary strings. Each string includes one or more production rules for an L-system. Production rules apply to an axiom a number of iterations or until the string contains only terminals. The resulting string converts into a structural specification for a network. Back propagation then applies to the network for a given task domain. A sum of the squares of the errors then acts as the fitness function for the evolutionary process. The example used in early works was that of an XOR gate.

In both Boers et al. and Kitano the L-system was only concerned with topology, as opposed to also evolving the weights. Further work did incorporate weights to L-system but at a rudimentary level (i.e binary weights) [82]. These works inspired others to use properties of L-system while incorporating greater robustness for topology and weight evolution; Cellular Encoding (CE) is an example of one of those approaches.

2.2.2 Cellular Encoding (CE)

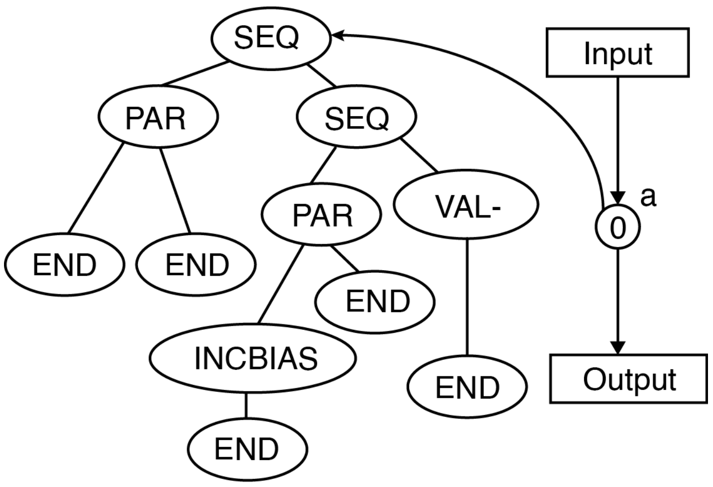


Figure 2.10: **Cellular Encoding of a neural net for XOR**. Figure adapted from [83].

CE takes inspiration from cell division in living organisms [81, 83]. Like L-system, the network is representing a set of directions for its construction, rather than as a direct specification. The philosophy of the approach is to imitate the interactions that occur among proteins and cells in a developing embryo. A grammar tree represents the developmental process. The process starts with a single node. Then, by the use of cell division, the tree branches out, adding new cells. Cell types dictate how the neural network constructs. By curating cell types, restriction in the structure can leverage domain exploits; in which specific network topologies are beneficial in a domain. The cell types below were those in the original work:

- **A division cell** - which creates two cells from one, further rules can be applied to the relationship the mother cell has to offspring.
- **A value cell** - modifies the link in the link register to +1 or −1, which then moves the pointer.
- **An unary cell** - which deletes the link stored in the link register.
- **The waiting cell** - which makes the cell wait for its next rewriting step.
- **The end cell** - which makes a neuron from the current cell and stops rewriting.

Crossover and mutation operation apply according to the common genetic programming paradigm. Mutation takes a random node on the tree and replaces it with a different instruction. Then, crossover cuts a subtree from one parent tree and replaces a subtree from the other parent tree; the result is an offspring tree. The subtrees exchanged during recombination are randomly selected.

Introductory work showed how cellular encoding could replicate an XOR gate. But since then it has been shown that CE does not compete when compared to other NE methods in bench marking tasks; including direct encoding [75, 55]. AGE is among those which beat out CE. Like CE, AGE takes inspiration from gene regulatory networks; the networks formed by genes that send signals back and forth through their protein products.

2.2.3 Analog Genetic Encoding (AGE)

AGE is an implicit method derived from the observation of biological genetic regulatory networks [83, 55]. First, AGE relies on an explicit range of characters to construct the chromosome. Commonly, the ASCII alphabet is used for this range. Chromosomes represent the genome of a neural network. Genome's size is a variable defined before run time. For example, in Dürr et al. [55] the initial genome was sized between 500 and 800 characters, with each character being a possible of 26 unique characters of the ASCII uppercase alphabet. Within the chromosomes there are distinct sequences of characters called tokens. The token characters are defined before run time. Each token defines a neuron (neuron token) or type of neural connection (terminal token). Characters between a neuron token and the first terminal token, as well as those between each terminal token, are terminal sequences. Terminal sequences influence the neural connection relationships. Characters between the final neural connection and a new neuron token are non coding characters (i.e are not used in the phenotype).

During phenotype construction, the genome is sequentially inspected. Each neuron token creates the appropriate neuron in the network. Terminal sequences group in sequential pairs and their similarity recorded; this is the alignment score. Via a specified threshold, the alignment score determines whether the paired nodes connect. Provided a successful connection takes place, the alignment score determines connection weight. Although it can be specified, the first sequence following a neuron token is the neuron output and the second sequence the neuron input.

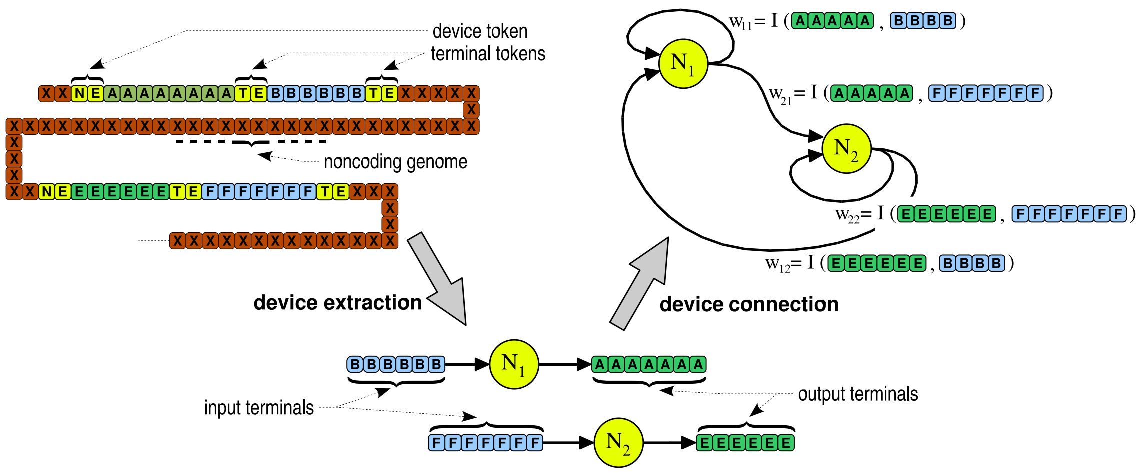


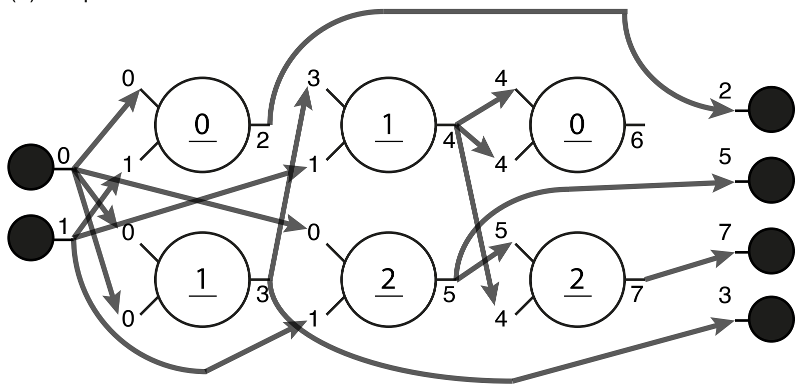
Figure 2.11: **AGE development process**. Figure duplicated from [55].

AGE was benchmarked on the double pole balancing problem with no velocity, beating NEAT and establishing a new state-of-the-art result at the time [55]. As well as finding success in other areas like electronic circuits [145, 146]. Despite this, AGE is restricted to low dimensionality domains. As with direct encoding, the genome type length is tied to the size of the network. Task domains with greater complexity will require larger inputs and, for this method, a large genome.

2.2.4 Cartesian Genetic programming (CGP)

CGP is a popular and efficient graph based form of Genetic Programming [154]. A powerful aspect of CGP is its representation of graphs coupled with a high degree of genetic redundancy. These graphs are represented as a two-dimensional grid of computational nodes where user's define the number of columns and rows. In the

(1) Graph



(2) Encoding of graph as a list of intergers (i.e. the genotype)

0 0 1   1 0 0   1 3 1   2 0 1   0 4 4   2 5 4   2 5 7 3

(3) Function Genes

0 + Add the data presented to inputs
1 - Subtract the data presented to inputs
2 * Multiply data presented to inputs
3 / Divide data presented to inputs (protected)

Figure 2.12: **CGP graph representation**. Figure adapted from [153].

genotype, each gene contains their input, output, and action within the graph. Each action is user defined and placed into a function look-up table. The associated integer values to those functions are called function genes. The input and output integers are called connection genes. The genotype is a fixed length; however, the amount of computational nodes can vary from zero to the number of nodes defined in the graph. When decomposing the genotype to phenotype, some nodes may be ignored. This occurs when the node's outputs do not contribute to the output data; these are 'non-coding' genes. The phenotype is the graph representation. Each node's maximum amount of inputs is relative to total inputs of the graph (arity). Each node contains an integer address for their output.

Each address sequentially follows on from the number of inputs. Nodes in the same columns cannot be connected to each other. In the classic implementation, the graphs are feed forward, therefore nodes may only have its inputs connected to either input data or the output of a node in a previous column. However, there is a level-back parameter which controls how many columns back a node can connect from. CGP's initial implementation was associated with digital circuit design [155, 236]. However, the graph structure allows a simple transition to ANN, as seen in Cartesian Genetic Programming of Artificial Neural Networks (CGPANN) [116, 235]. The function genes are swapped for activation functions, such as sigmoid or hyperbolic tangent; the selection of which can have significant impact on the task [237]. Additional genes are added for weights. Switch genes are added as binary gates to whether the connection is active. During mutation, weights are replaced by a new random real number. Switch genes change to the opposite activation. Therefore, topological features can be added and removed by mutating functions, connections, weights, switches, and outputs. CGPANN has been extended to allow for recurrent connections (RCGPANN) to be effective on tasks that require recurrency (i.e the double pole balancing task without velocity [116] and forecasting [238]). CGP has proven competitive on standard benchmarks [235] and general game playing [256], as well as versatile in tasks such as Wumpus [115], checkers [114] and maze solving [88].

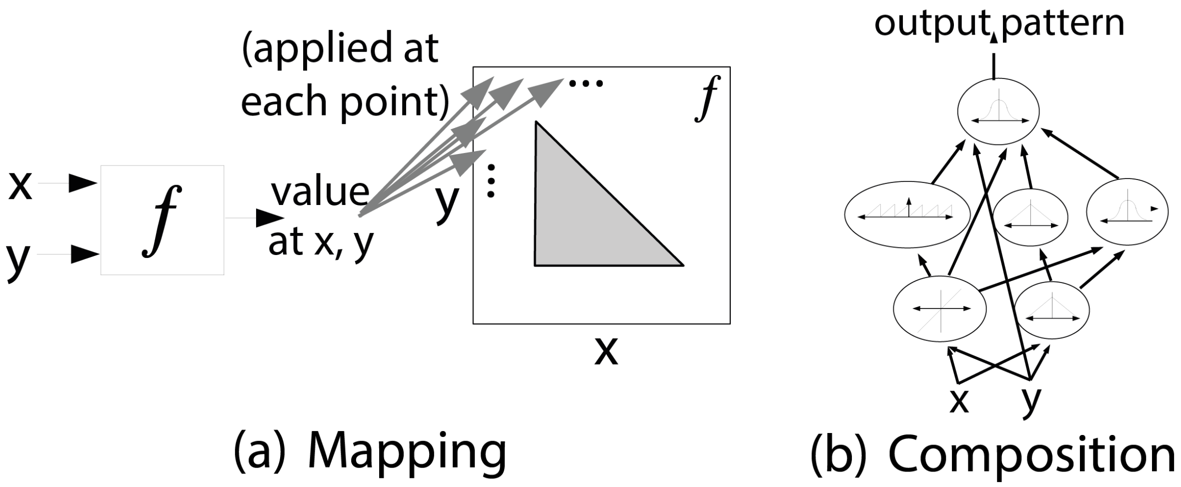


Figure 2.13: **CPPN encoding**. (a) The function f takes arguments x and y, which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of f, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is f. (b) The CPPN is a graph that determines which functions are connected. The connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. Figure adapted from [43].

2.2.5 Compositional Pattern Producing Networks (CPPN)

CPPN aims to achieve development encoding without development [211]. CPPN's structure is near-identical to that of a neural network, the difference being in activation functions. CPPN uses a variety of evolvable activation functions rather than limited to Sigmoid and Gaussian. With only the activation functions to consider, previous direct encoding integrates seamlessly. This is why NEAT is the default method for CPPNs [43, 203, 91, 34].

Each activation function replicates a biological representation; symmetry (e.g Gaussian function), repetition (e.g sine function), Imperfect Symmetry etc [214]. By restricting activation functions the CPPN can bias towards desired patterns. For example, the use of polar coordinates and sine functions (repetition) flower patterns can form [191]. The most notable example of CPPN's ability to generate procedural nature inspired content is PicBreeder [203].

Picbreeder is an online service that allows users to collaborate on the formulation of images via a CPPN. Each generation, fifteen images are produced and presented to the user. Then, the user selects one or more images as parents for the next generation, as well as a mutation rate. Users can also evolve other user's outputs; essentially piggybacking off their evolutionary exploration. Picbreeder is the visual affirmation that CPPN replicates natural biological patterns by creating human recognisable structures from random initial images. Examples of these structures include: cars, planets, and skulls.

A point of contention is the validity of these images as natural structures as the result is subjective to each user. However, various PicBreeder images have been recognised and sensibly classified by pre-trained deep neural networks (DNNs) for the classification of real world objects [60]. This result suggests the patterns CPPN generates do accurately replicate forms of natural biological phenomenons. Picbreeder benefits from the exploitative nature of human selection. Without curated evolution, and instead a fitness function of similarity to Google images, images become more akin to genetic art than recognisable natural evolutionary structures [7]. Objective based algorithms were also examined in PicBreeder. When the target output were previously generated PicBreeder images, all runs either failed or took substantially longer than its interactive evolutionary counterpart [258]. There have been implementations in which less explicit techniques are used to allow interactive evolution, such as in Galactic Arms Race (GAR). New weapons in GAR generate based on user preference, which is how long a user spends using that weapon [91].

2.2.6 Hypercube-based NEAT (HyperNEAT)

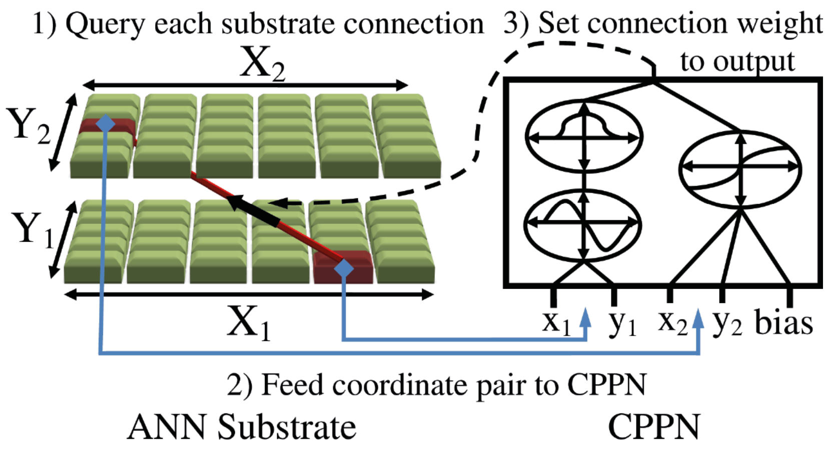

1) Query each substrate connection
3) Set connection weight to output
X2
Y2
Y1
X1
x1 y1 x2 y2 bias
2) Feed coordinate pair to CPPN
ANN Substrate
CPPN

Figure 2.14: **A CPPN connectivity example for HyperNEAT.** A grid of nodes, called the ANN substrate, is assigned coordinates. (1) Every connection between layers in the substrate is queried by the CPPN to determine its weight; the line connecting layers in the substrate represents a sample such connection. (2) For each such query, the CPPN inputs the coordinates of the two endpoints, which are highlighted on the input and output layers of the substrate. (3) The weight between them is output by the CPPN. Thus, CPPNs, whose internal topology and connection weights are evolved by HyperNEAT, can generate regular patterns of connections. Figure duplicated from [245].

HyperNEAT exploits geometric regularity in task domains via the use of CPPN to encode weight matrices [221]. The results have shown good performance on a wide variety of problems; multi-agent solutions [44], simulated locomotion [32, 33, 31], physical locomotion [130], autonomous robot cars [54], checkers [66] etc.

HyperNEAT uses two networks, a CPPN and a substrate. The CPPN encodes the connectivity pattern of the substrate. The substrate is an ANN whose nodes simulate a coordinate system to represent the topology. The name was chosen to be verbally distinguished from the CPPN which has its own topology [65], With the use of the CPPN, HyperNEAT can exploit geometric properties, such as regularity and repetition. As such, the substrate includes information to allow the CPPN to be geometrically aware. So, the substrate configuration is dependent on the task domain. For example, the state-space sandwich is the configuration used in checkers and quadruped locomotion. This is a single two dimensional sheet of neurons fully-connected to another two-dimensional sheet. Another quadruped substrate divides each control of a leg into substrate modules [188]. HyperNEAT is still able to encode traditional ANNs (i.e two-dimensional substrates) [42], as well as nontraditional evolutionary networks like a Convolutional Neural Network (ConvNet) [244, 202]. Each layer has Y representative and each node has a X representative. This allows connective CPPNS to use spatial information.

In Hausknecht et al. [94], HyperNEAT was assessed on general game playing with atari 2600 games. Three different state representations were used; object, seeded noise and pixel. In the comparison with other top direct encoding methods, HyperNEAT was the only approach to exploit the relationship of the game with raw pixel colours. Although Deep Mind made a larger impact on this task later, HyperNEAT was the first system for which direct pixel-to-action atari results were reported.

Though, the direct encoding methods did out perform HyperNEAT on those other representations. Yet, this is due to the low dimensionality of the representation; which direct encodings are superior at. Still, it is a testament to HyperNEAT's ability to adapt to general-purpose problems.

Due to the separation of CPPN and substrate, the underlying mathematical relationships can be retained even when the substrate changes. This allows the substrate resolution to fluctuate to appropriate tasks without further evolution, as seen in examples in which the substrate was scaled [65, 246]. Therefore, HyperNEAT can generalise significantly more effectively [66].

HyperNEAT has a weakness for irregular task domains. Clune et al., [34] demonstrates how HyperNEAT reacts to solving problems as the task irregularity decreases. The conclusion is that performance decreased as problem regularity decreased. Direct encoding (P-NEAT) out performed HyperNEAT but only when the task is relatively irregular. This issue, among others, has caused numerous extensions to HyperNEAT.

Extensions

Due to HyperNEAT's popularity it has been used as a foundation to build extensions that incorporate new and old ideas, while leveraging HyperNEATs proven versatility and performance. For example, Hybridized Indirect and Direct encoding (HybrID) takes the already established benefits of direct encoding and combines them with indirect encoding. Indirect encoding exploits available regularities, then switches to direct encoding to account for irregularities. The encoding switches after a number of generations. In testing domains, HybrID outperformed HyperNEAT in every run [34]. However, the significance is only apparent when the problem domain has a certain amount of regularity. Also, pathways bias the exploration to what indirect encoding finds initially desirable. A potential solution to this is R-HybrID [207]. The switch from indirect to direct encoding are under evolutionary control. In three benchmark tasks R-HybrID outperformed HyperNEAT, NEAT, and HybrID. An issue to consider is how direct encoding scales to large substrates. It may be too computationally intensive for current machines to individually change each weight. Future work may see techniques to incorporate direct encoding into an abstracted set of parameters, without dealing with each individually.

Adaptive HyperNEAT employs lifetime learning into the CPPN to encourage the discovery of local learning rules. The vague formulation of local learning requires (1) learning to depend on local information associated with the pre and post-synaptic neurons; and (2) learning ought to depend on the correlation between the activities of these neurons, yielding a spectrum of possibilities on how these correlations are computed and used to change the synaptic weights [9]. This is inspired by how biological brains can adapt and learn from past experience. Three models are proposed in this work. The iterated model provides inputs for activation of the pre-synaptic and post-synaptic neuron, and the current weight as input. The ABC Model produces three CPPN outputs that each control synaptic plasticity during the lifetime of the agent, as well as a learning rate. Finally, Plain Hebb which only includes the learning rate as in output. The iterated and ABC model was able to solve a task with Nonlinear Reward Signature, where Plain Hebb failed. We can deduce then that a standard HyperNEAT configuration would not be adequate to the task. In the Iterated model, the CPPN is required at every clock tick to update the ANN weights as the inputs are needed to affect plasticity. However, it is acknowledged, this is a more computationally intensive ask. The substrate used in this work was only a few neurons in a two layer network. But, an appeal of HyperNEAT is the large structures that can efficiently be encoded. This method may not be practical for those tasks.

Evolvable-substrate HyperNEAT (ES-HyperNEAT) allows the substrate design to be dynamic through the evolutionary process, like NEAT before it. In the original HyperNEAT implementation the placement of nodes in a substrate is decided by the user and these placements play an important role in the ability to exploit geometric symmetry in a task domain. Therefore, HyperNEAT asks the user to choose an appropriate substrate to best fit the environment, however this may not be a clear choice. The example used for ES-HyperNEAT is the difficulty defining the best placement for sensors and effectors to relate to a domain geometry as well as the appropriate number of hidden nodes. Rather, ES-HyperNEAT leaves the substrate undefined prior to evolution, only the inputs and outputs are defined. The placement of nodes is then the responsibility of the CPPN. Based on implicit information in an infinite-resolution pattern of weights, the CPPN is able to produce patterns of possible connections, including node positions. Nodes which contribute to the overall results (i. connect in some form to an input are output) remain in the substrate, while the other connections are pruned. A new important factor introduced with ES-HyperNEAT is representing regions of varying density in the substrate which cluster due to the CPPN's node geometric position. As pointed out this is an abstracted feature of neurons in the brain. As the architecture is connected to the CPPN the substrate is able to exploit complex regular patterns. This approach was demonstrated to be more efficient than the original HyperNEAT at solving a maze navigation task and a task that required switching sensors [185, 186]. These extensions can also be stacked together, see ES-HyperNEAT and Adaptive HyperNEAT [187].

2.2.7 Deep Neuroevolution

Deep neuroevolution refers to the evolution of deep neural networks, which typically have (many) more than two hidden layers. Due to the large number of parameters it is infeasible for direct encoding. Deep neuroevolution uses a simple random number generator for weight parameters and a genome of seed values for the number generator. Generating the vector of parameters can be achieved by applying each seed in chronological order. Then, a conventional evolutionary process can take place on the population (i.e mutation, elitism). Each mutation, a new seed is added to the end of the sequence. A formal mathematical definition can be seen at figure 2.15.

So, the genome can only grow as long as the number of generations and, as demonstrated, can feasibly generate up to 4 million parameters. Elitism is employed to retain the E most fit individual(s) into the next generation, un-mutated. Parents are selected

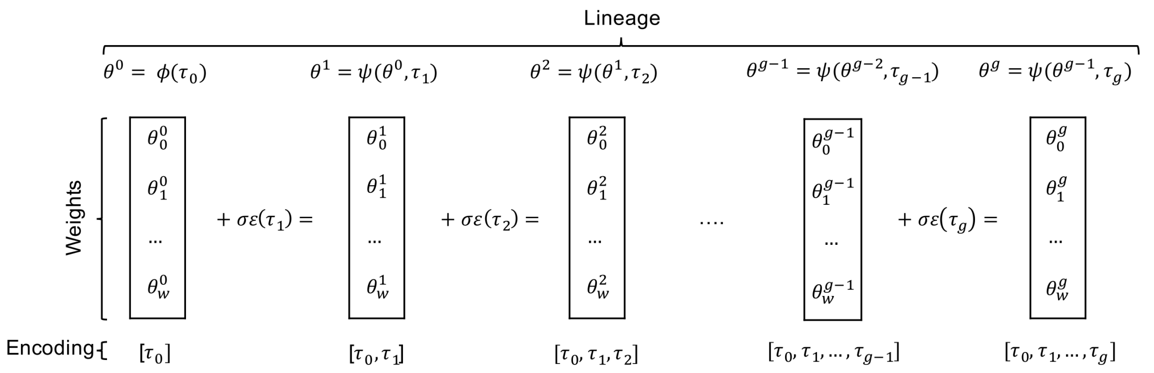

Lineage
$\theta^0 = \phi(\tau_0)$
$\theta^1 = \psi(\theta^0, \tau_1)$
$\theta^2 = \psi(\theta^1, \tau_2)$
$\theta^{g-1} = \psi(\theta^{g-2}, \tau_{g-1})$
$\theta^g = \psi(\theta^{g-1}, \tau_g)$
Weights
$+\sigma\varepsilon(\tau_1) =$
$+\sigma\varepsilon(\tau_2) =$
$+\sigma\varepsilon(\tau_g) =$
Encoding
$[\tau_0]$
$[\tau_0, \tau_1]$
$[\tau_0, \tau_1, \tau_2]$
$[\tau_0, \tau_1, \dots, \tau_{g-1}]$
$[\tau_0, \tau_1, \dots, \tau_g]$

Figure 2.15: **Visual representation of the Deep GA encoding method**. From a randomly initialised parameter vector θ^0 (produced by an initialisation function ϕ seeded by τ_0), the mutation function ψ (seeded by τ_1) applies a mutation that results in θ^1 . The final parameter vector θ^g is the result of a series of such mutations. Recreating θ^g can be done by applying the mutation steps in the same order. Thus, knowing the series of seeds $\tau_0 \dots \tau_g$ that produced this series of mutations is enough information to reconstruct θ^g (the initialisation and mutation functions are deterministic). Figure duplicated from [226]

uniformly at random from the T most fit individuals in a generation and mutation applied to create new offspring; this process repeats until the next generation's population is filled.

Deep ConvNets have predominantly been trained by SGD, achieving state of the art results. NE has aided in topology building [152, 263, 244] and feature extraction [202, 10] but, until recently, showed little promise in training very large neural networks. Such et al. [226] and Salimans et al. [194] demonstrated that genetic algorithms and evolutionary strategies are competitive alternatives for training large neural networks, including deep ConvNets, on RL tasks. Both have been used on RL benchmarks using the Arcade Learning Environment (ALE) and MuJoCo, producing competitive and state-of-the-art results. Such et al. [226] used a very simple GA, with no recombination.

Their results gave a proof of concept and suggested that techniques such as crossover [62], exploiting regularities [34] and diversity mechanisms [167, 165] should be explored in future extensions. Due to deep neuroevolution utilising ConvNets to achieve state-of-the-art results in its introductory paper [226], the next subsection will cover ConvNets and how it differs from the conventional feed forward discussed up until now.

Convolutional Neural Network (ConvNet)

ConvNets were originally proposed in LeCun et al. [129] for handwritten digit recognition. They proved successful also in speech recognition [128], object detection in natural images [241] and face recognition [127]. The basis of the modern ConvNet architecture was introduced in Yann et al. [265] with LeNet-5. LeNet-5's success comes from deriving higher-level features from identified lower-level ones; this is achieved via local connections, shared weights, pooling, and the use of layers. ConvNet can consist of convolution layers, pooling layers, non-linearity and a fully-connected layer.

Kernel convolution is a process where a small matrix of numbers (kernel) passes through a matrix or tensor and transforms it depending on the weights within the kernel. Convolution extracts spatial or temporal features and many kernels can be used to extract various features from an image. The following equation expresses kernel convolution:

$$G[m,n] = \sum_{j} \sum_{k} h[j,k] f[m+j, n+k] \tag{2.4}$$

where f represents the input, h the kernel, m the rows and n the columns of the result matrix. Once transformed, a non-linear activation function is applied to the outputs. It is then common for ConvNet's to use pooling layers which create an invariance to small shifts and distortions, pooling reduces the size of the tensor by down-sampling. How pooling down samples can be tuned to the task domain; for example, average pooling calculates the average for each kernel patch of the feature and is used when data retention is necessary over multiple layers, whereas max pooling, carries through the largest value in each kernel patch of the feature and therefore retains only the most present feature in the patch. After multiple sequences of convolution, non-linearity and pooling layers, the rows and length of features reduce while the layer size increase; this can be calculated with the following:

$$[n, n, n_c] * [f, f, n_c] = \left[\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, n_f \right] \tag{2.5}$$

where n represents tensor input length or height, f represents filter size, n_c number of layers in input tensor, p is padding, s is stride, n_f the number of kernels. The final fully connected layer represents a traditional ANN, where the tensor is flattened to a one dimensional matrix and fully connected to an output layer or layers. In a classification task, for example, the fully-connected section will determine how features in the previous layer contribute to each class.

Despite these initial successes, ConvNets' greater popularity only came to fruition with advances made in core computing systems. The use of graphical processing units allowed AlexNet [121] to train a deeper and wider ConvNet. In the challenging ImageNet competition, AlexNet achieved state of the art results. Various other advances were introduced in this architecture, including the use of dropout to reduce overfitting and the Rectified Linear Unit (ReLU) to improve training times. Since then, ConvNets have been structured in many different ways. The work in Mnih et al. [157] omitted the pooling layers to retain spatial information. Residual Networks (ResNets) have employed more than 100 layers to improve performance on visual recognition tasks [96]. GoogLeNet's Inception module uses multiple kernels, at different sizes, on the same input, to abstract features from different scales [229]. DenseNet connects each layer to every other layer in a feed-forward fashion which strengthens feature propagation [104].

2.3 Diversity

The basis of CNE selection pressure is to produce fitter solutions while discarding those with lower fitness; as a result, populations converge in the search space. Then, solutions may stall as GAs struggles to effectively explore the search space. These types of problems can be expressed as bootstrap or deception [208]. Consequently, fitness functions may not provide a selection pressure in a task domain. This makes solutions too difficult for the evolutionary system to discover directly [71]. Individuals may perform poorly and drift in an uninteresting region of the search space (i.e the bootstrap problem). Or, promising initial performance leads the population to a local optima, (i.e deception) [253]. However, by preserving diversity within a population, evolution can benefit from greater exploration. It is one of the most common methods to improve the behaviour of GAs with demanding fitness functions [168]. Diversity also promotes quick exploration of the search space. If a population converges, the mutation operator is the only viable solution to continue exploration; this progresses very slowly [163]. Yet, a diverse population can utilise the crossover operation to make larger traversal strides in a short amount of time. Long have techniques been devised to introduce diversity into NE [70, 141]. The following sections will examine the most popular types.

2.3.1 Crowding

Crowding promotes diversity by replacing existing individuals with new genetically similar ones [48]. Each generation, selections of the population are evaluated to discover parents. With the production of an offspring, another sub-population is defined by the Crowding Factor. The offspring replaces the most genetically similar individual in the sub-population. Deterministic Crowding improves upon the original by allowing competition between parent and child of identical sub-populations [140]. If offspring are in a genetically similar range to their parents and have a higher fitness, the offspring replaces the parent. Therefore, two sets of tournaments between both parents and both offspring are possible. The following pseudocode describes this:

```
Algorithm Deterministic Crowding
  for $g \in G$ do                                             ▷ Repeat for $G$ generations
      Select 2 parents, $p_1$ and $p_2$, randomly, no replacement
      Cross them, yielding $c_1$ and $c_2$
      Apply mutation / etc., yielding $c_1'$ and $c_2'$ (optional)
      if $\left[d\left(p_1, c_1'\right) + d\left(p_2, c_2'\right)\right] \leq \left[d\left(p_1, c_2'\right) + d\left(p_2, c_1'\right)\right]$ then
          if $f\left(c_1'\right) > f\left(p_1\right)$ then
              replace $p_1$ with $c_1'$
          if $f\left(c_2'\right) > f\left(p_2\right)$ then
              replace $p_2$ with $c_2'$
      else
          if $f\left(c_2'\right) > f\left(p_1\right)$ then
              replace $p_1$ with $c_2'$
          if then
              replace $p_2$ with $c_1'$
```

Another extension is probabilistic crowding [150], where fitter individuals do not necessarily win over weaker individuals. The winning individuals is proportional to their fitness based on a probabilistic formula [140]:

$$p_x = p(x) = \frac{f(x)}{f(x) + f(y)} \tag{2.6}$$

where f is the fitness function and x and y are two similar individuals that have been picked to compete. The original crowding implementation has shown to be limited in multimodal function optimisation. Crowding fails to consistently maintain more than two peaks of a multimodal function [141]. This was attributed to stochastic errors in the replacement causing genetic drift; the loss of alternative solutions due to random fluctuation. Deterministic Crowding reduced the replacement error, which in turn, allowed maintenance of multiple peaks. Further studies showed that deterministic crowding aided in locating superior single solutions [92, 175]. By maintaining many sub solutions, greater exploration of the search space is promoted. Probabilistic Crowding furthers this idea by the exploration of less fit solutions.

2.3.2 Fitness sharing

Fitness sharing arranges the population into sections in the search space based on a metric of similarity and penalises those in higher clustered areas, thus forcing the GA to maintain diversity within the population [70]. This method is used to attempt to find all maximas in a given multimodal function. The inspiration for this method is species in nature occupy the same environment and therefore have to share resources. But, those individuals in the environment may want to spread out to seek more resources and higher rewards; thus the devaluation of fitness in close individuals. Similarity of individuals can be established via the genotype, phenotype or a combination of the two; the original work focused on the phenotype. To determine fitness sharing, two parameters are needed; the objective fitness and the niche count. The objective fitness is calculated via Goldberg's ranking [69] which provides equal reproductive potential for non dominated individuals. This method separates individuals in the population into layers and ranks them depending on their dominance in the search space. The least dominant are assigned rank 0 then removed from contention. The next set of non-dominant individuals are found to be given rank 1 etc. This is repeated until the population is suitably ranked. Next, the niche count provides an estimate to crowd together individuals into a niche. This requires a user defined niche radius to estimate the expected minimum separation between solutions. This particular function has been criticised for being domain specific, requiring prior knowledge [180, 112]. A simple approach to calculate an individual's shared fitness is [70]:

$$f_i' = \frac{f_i}{m_i} \tag{2.7}$$

where f_i is the shared fitness of an individual (i) and m_i is the niche count.

2.3.3 Incremental Evolution

Incremental evolution is a term that describes a change in environment or reward function to incrementally guide evolution through increasingly more difficult tasks. The

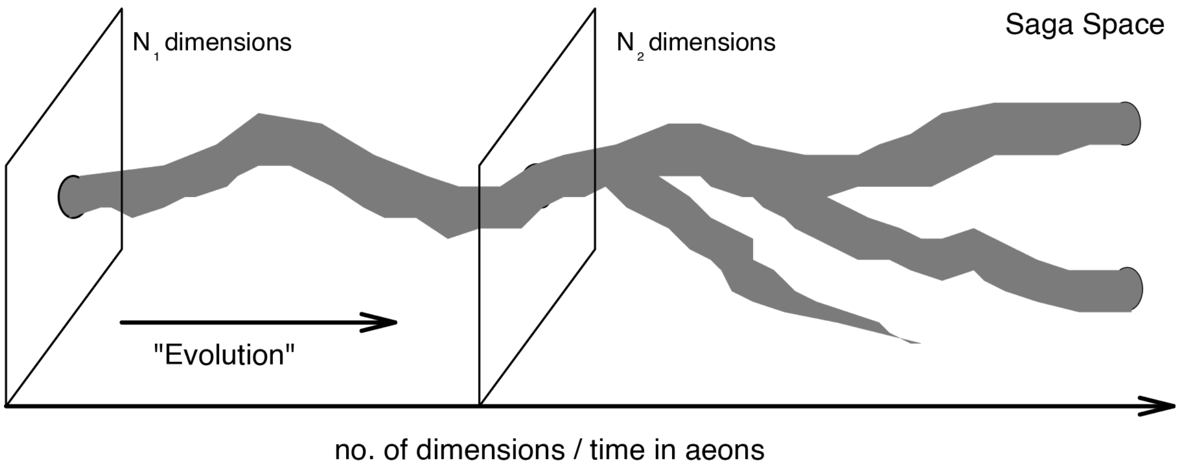


Figure 2.16: **Illustration of SAGA space**. The progress of the always compact course of a species the z axis indicates both time and the loosely correlated number of dimensions of the current search space The x and y axes represent just two of the current number of dimensions The possibility of splitting into separate species and of extinction are indicated in the sketch.. Figure duplicated from [89].

first inspirations for this technique were laid out in Inman Harvey's Species Adaptation Genetic Algorithms (SAGA) framework [89, 90]. The framework came from the observation that in nature evolving populations are highly converged genetically and this would be true at every point in history. So, for 4 billion years there has been the pathway of a population changing from a single cell to complex creatures. Therefore, complex individuals are linked via a continuous chain of viable ancestors to the origin. In an artificial evolutionary example, solutions may be unattainable due to lack of viable pathways from the origin. Therefore, incremental evolution breaks the task down into simpler tasks and, once solved, moves individuals into more challenging tasks until it is tasked with the goal itself; guiding individuals through the search space.

Gomez demonstrated the effectiveness of incremental evolution in a pursuit-evasion simulation by gradually increasing the speed of prey when the ANN controlled the predators [71]. Within this implementation, an agent is placed in the centre of the grid world and the prey is placed in a random position just within the agent's sensor range. The agent and prey alternate in taking an action at each time step until either the prey has been captured or a maximum number of time steps has been reached. The population first evolved on the task $E_0^{0.0}$ (i.e., capturing a stationary prey within its sensory range). Next, an evolutionary process takes place based on the successful individual's genotype (Δ). After each successful run of the task the environment increases the number of steps the prey can take ($0-4$) and at what speed (0.0 to 1.0 in four steps). This sequence forces the agent first to develop its memory and then to learn to deal with a fast-moving prey. This is expressed via [71]:

$$E_0^{0.0} \xrightarrow{\Delta} E_2^{0.0} \xrightarrow{\Delta} E_3^{0.0} \xrightarrow{\Delta} E_4^{0.0} \xrightarrow{\Delta} E_4^{0.3} \xrightarrow{\Delta} E_4^{0.6} \xrightarrow{\Delta} E_4^{0.8} \xrightarrow{\Delta} E_4^{1.0} \tag{2.8}$$

Other examples include, work from Bongard which demonstrated how robots that grow from anguilliform into legged robots then lose the anguilliform body later in evolution, performed more rapidly and more robustly than robots without this transition [18]. Paired Open-Ended Trailblazer (POET) [248] trains 2D bipedal walkers on an indefinite environment of increasing complexity. POET achieves behaviours that were unreachable in any conventional way.

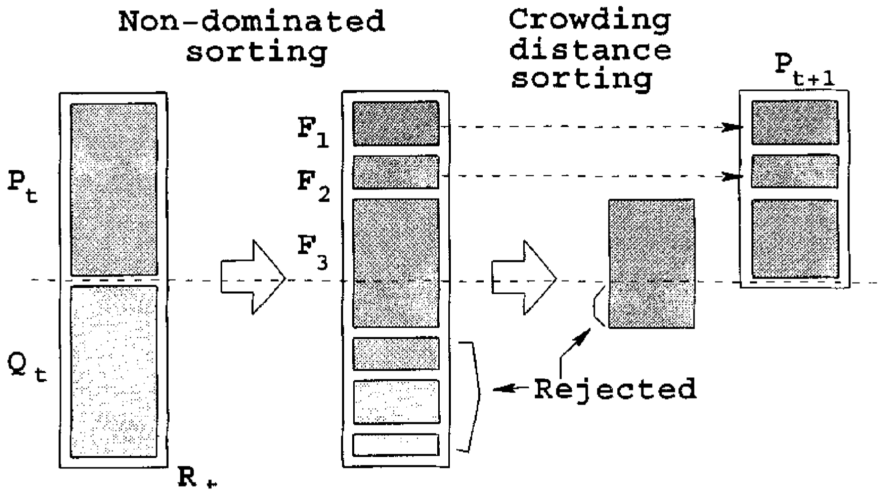


Figure 2.17: **NSGA-II procedure.** Figure duplicated from [51].

2.3.4 Multi-objective evolutionary algorithms (MOEAs)

MOEAs are designed for multi-objective domains in which a solution may require trade-offs. When in a domain that has multiple objectives, this causes multiple optimal solutions (known as pareto-optimal solutions), as opposed to a single solution. Without a leaning to any one solution, no solution can be superior to another, therefore it is evolution's role to find many pareto-optimal solutions within a single run. MOEAs have two ideal goals: (1) converge on a set of solutions which lies on the pareto-optimal front and (2) diversify enough to represent the entire range of the pareto-optimal front [50]. The first practical algorithm was Shaffer's Vector Evaluated Genetic Algorithm (VEGA) [197]. VEGA ran independent selection cycles according to each objective. Each objective is carried out on parts of the population to fill a mating area. Then, via crossover and mutation the population gains offspring from different subgroups. VEGA suffers from biases towards certain pareto-optimal solutions, which in turn converges the entire population towards the individual optimum regions after a large number of generations. Goldberg provided suggestions based on a non dominated sorting procedure to improve upon Schaer's ideas. Non-dominated sorting ranks individuals in how they dominate others; for an individual to dominate another they have to (1) not be objectively worse in any objective than the other and (2) at least one objective is superior over other individuals. This repeats until all non-dominated individuals have been ranked. This is detailed in pseudocode on the next page. Non Dominated Sorting Genetic Algorithm (NSGA) [210] implements Goldberg's suggestion. NSGA showed an ability to maintain a stable and uniform reproductive potential across non dominated individuals, overcoming VEGA's shortcomings. The next major iteration to this set of algorithms was the inclusion of elitism; the most popular of which is NSGA-II [51]. Before the non-dominated sorting, NSGA-II creates an offspring population with the previous population as parents. Usual genetic operators are applied, then the two populations are merged to be double the size of the original population. Next, non-dominated sorting applies. Due to the larger population size, those below the original population size are discarded; with those in the final ranking sorted via highest diversity before discarding the remaining individuals. A real-world example of an application is spacecraft trajectory optimisation with objectives such as; maximising the delivered payload; minimise the time of flight and maximising heliocentric revolutions [37].

Algorithm fast-non-dominated-sort(P)

```
for all $p \in P$ do
    $S_p = \emptyset$
    $n_p = 0$
    for all $q \in P$ do
        if $(p \prec q)$ then                    ▷ If $p$ dominates $q$
            $S_p = S_p \cup \{q\}$               ▷ Add $q$ to the set of solutions dominated by $p$
        else if $(q \prec p)$ then
            $n_p = n_p + 1$                      ▷ Increment the domination counter of $p$
    if $n_p = 0$ then
        $p_{\text{rank}} = 1$
        $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$
$i = 1$                                          ▷ Initialise the front counter
while $\mathcal{F}_i \neq \emptyset$ do
    $Q = \emptyset$                              ▷ Used to store the members of the next front
    for all $p \in \mathcal{F}_i$ do
        for all $q \in S_p$ do
            $n_q = n_q - 1$
            if $n_q = 0$ then
                $q_{\text{rank}} = i + 1$
                $Q = Q \cup \{q\}$
    $i = i + 1$
    $\mathcal{F}_i = Q$
```

2.3.5 Novelty Search (NS)

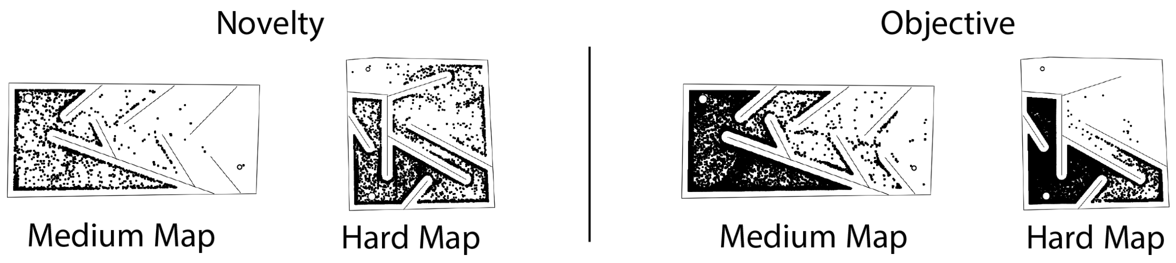


Figure 2.18: **Novelty and objective fitness based search on medium and hard maze environments.** Each maze depicts a typical run, stopping at either 250, 000 evaluations or when a solution is found. Each point represents the end location of a robot evaluated during the run. Novelty search is more evenly distributed because it is not deceived. Figure adapted from [190].

NS was conceived for deceptive domains in which reward-based optimisation mechanisms converge to local optima; this has been demonstrated in maze tasks [166], but also less obvious applications like bipedal walking [131]. NS exploits the observation that in non-human populations diversity in behaviours provides greater success in problem-solving [15]. NS ignores the reward function during evolution and instead focuses on agents performing new, unseen behaviours. To assess what is a novel behaviour, a novelty metric is required; good behaviour characterisation has shown to be essential to NS's success [118, 169]. In the hard maze task, the objective fitness would favour bots that end up closer to the goal. With NS, the novelty metric behaviour is defined by the cartesian location of the robot in the maze. Then, the novelty metric calculates the average distance between it and its k-nearest neighbours; where k is a fixed parameter that is determined experimentally. The nearest neighbours calculation takes into consideration individuals from the current population and from the permanent archive of novel individuals. The archive consists of past individuals with highly novel behaviours, that is, those above some minimal threshold p min of novelty. The archive also mitigated backtracking to previously discovered behaviours. The following equation expresses the behaviour metric [166]:

$$\rho(x) = \frac{1}{k} \sum_{j=0}^{k} d_b \left(x, \mu_j\right) \tag{2.9}$$

where k is a user-defined parameter and $_j$ is the j-th nearest neighbor of x with respect to the distance d_b. NS was first demonstrated with NEAT which continually ascends to new levels of complexity, which in turn allows more novelty. Yet, novelty can be used in place of any NE method which already uses an objective fitness function; notable examples are HyperNEAT [164] and Deep Convolutional Networks [36]. NS has inspired implied extensions; Surprise Search [79] rewards exploration and divergence from the expected behaviour, and Evolvability Search [149] selects individuals with a greater potential for diversity, rather than diversity itself. Search based on pure novelty has shown not to scale well in tasks with large feature spaces [132, 38] which is addressed in Quality Diversity (QD) algorithms.

2.3.6 Quality Diversity (QD)

Quality Diversity (QD) extends the early divergent search algorithms, like NS, by continuing to utilise diverging behaviours but also promoting an objective quality of those behaviours. QD differs from MOEAs as they are still ultimately driven toward

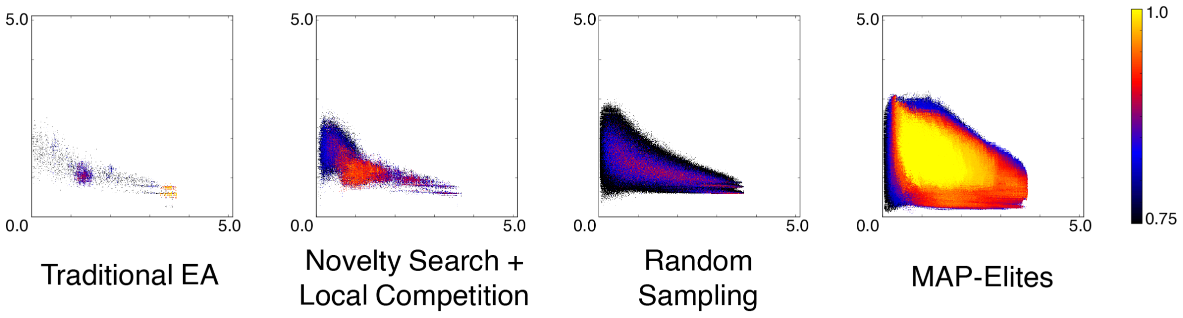


Figure 2.19: **Map-Elite heat map comparison of connection cost and modularity with other search techniques.** The x-axis is connection cost, the y-axis is modularity, and heat map colours indicate normalised performance. These maps show that MAP-Elites illuminates more of the feature space, revealing the fitness potential of each area. Figure adapted from [165].

specific objectives, which is intrinsically convergent. QD avoids the use of global fitness values as it pushes search to higher performing niches. Instead, QD aims to find all possible behavioural niches and present the most qualified candidate/s from that niche.

The first implementation of this type was NS with local competition [133]. This implementation uses NS to explore the search space based on sheer novelty, then species are formed depending on how close the species are. These individuals are then evaluated on a global fitness measure. The idea is to explore the merits of each niche rather than to exploit greedily only the best niches. Individuals are compared to their nearest neighbours in a niche and given a local competition score based on how many neighbours it outperforms. It is believed that this implementation produces more natural evolutionary dynamics through a gradual accumulation of functionally-diverse well-adapted individuals.

Later, Map-Elites [165, 39] was introduced which called itself a 'illumination algorithm'. MAP-Elites establishes a performance measure, which is a global fitness. Next, a user can add any number of variations of interest that define a feature space. Each dimension of feature variation is discretised based on user preference. MAP-Elites will then search for solutions for the highest performance measure with a variation of each interest. For example, if MAP-Elites was tasked with finding the morphology for a fast robot and the interests were size, weight, and energy consumption. MAP-Elites will search for the fastest robot that is tall, heavy, and efficient; the fastest robot that is tall, heavy, and inefficient, the fastest robot that is tall, light, and efficient, etc. The interests could be established directly in the genotype or may require evaluation in the phenotype while it performs. MAP-Elites describes the accumulation of interests as features, and each feature has a cell. Cells allow individuals to compete in niches to gain occupancy of the cell. There is no guarantee that all cells in the feature space will be filled, as individuals may not be found during search or they may not be possible. The pseudocode for Map-Elites can be found on the next page.

Like with previous divergent search algorithms, quality of solutions is dependant on the behaviour characterisation; how behaviours are calculated and compared. Studies on behaviour characterisation quality misalignment have shown that in simple domains the solution discovery is slower with passable solutions but in much more complex domains QD breaks down without the appropriate behaviour characterisation [178, 179].

Algorithm MAP-Elites Algorithm (Simple, Default Version)

```
($\mathcal{P} \leftarrow \emptyset, \mathcal{X} \leftarrow \emptyset$)        ▷ Create an empty, $N$-dimensional map of elites: {solutions $\mathcal{X}$ and their performances $P$}
for iter = $1 \rightarrow I$ do        ▷ Repeat for $I$ iterations
    if iter < $G$ then        ▷ Initialise by generating $G$ random solutions
        $x' \leftarrow$ random_solution()
    else        ▷ All subsequent solutions are generated from elites in the map
        $x \leftarrow$ random_selection($\mathcal{X}$)        ▷ Randomly select an elite x from the map $\mathcal{X}$
        $x \leftarrow$ random_variation(x)        ▷ Create x′, a randomly modified copy of x (via mutation and/or crossover)
    $b' \leftarrow$ feature_descriptor ($x'$)        ▷ Simulate the candidate solution x′ and record its feature descriptor b′
    $p' \leftarrow$ performance ($x'$)        ▷ Record the performance p′ of x′
    if $\mathcal{P}(b') = \emptyset$ or $\mathcal{P}(b') < p'$ then        ▷ If the appropriate cell is empty or its occupants's performance is ≤ p′,then
        $\mathcal{P}(b') \leftarrow p'$        ▷ Store the performance of x′ in the map of elites according to its feature descriptor b′
        $\mathcal{X}(b') \leftarrow x'$        ▷ Store the solution x′ in the map of elites according to its feature descriptor b′
return feature-performance map ($\mathcal{P}$ and $\mathcal{X}$)
```

2.4 Discussion

The preceding sections provide an overview of all the relevant techniques that were reviewed and considered for this work. The following section will discuss ideas on how these findings affect work related to motion planning. It has been identified that indirect and developmental encoding are most effective in high-dimensional task domains, while direct encodings are ideal for low dimensional domains. For motion planning, the dimensional size will be decided by the architecture and implementation. Artificial potential field implementations map the real space into a configuration space, a mapping of the geometry into discrete positions. This space is then transformed for robots to interact with. The obvious configuration space in a NE approach will be the representation of discrete spaces as neurons in an ANN. Next, the topology of this network.

To accommodate unknown and unforeseen future tasks, and also to refrain from biasing the search space, the motion planning network should be a versatile architecture. This means accommodating for; reactive behaviours, in which agents are navigating objects in a local space; deliberative behaviours, in which agents must traverse a long distance to obtain the goal; and a combination of the two. From an ANN perspective, each neuron in the environment should have a relationship with each other to achieve this range of behaviours. But, this would produce high dimensional search spaces as environment sizes increase. However, as Fekiac identifies, exactly how the network size contributes to solving a problem is not known when comparing between direct and indirect encoding; instead, putting greater emphasis on the regular and modular architectures [61]. Therefore, we do not yet have a grasp on how large of an environment direct encoding could accommodate or whether it is even suitable for small environments. For this reason, the initial experiments will focus on the scalability of each approach.

For the initial experiments NEAT and HyperNEAT have been chosen to cover direct and indirect encoding methodologies. It was established that on benchmark tasks, that if an appropriate topology was previously known, a fixed topology method would outperform NEAT, as was seen with SANE, ESP, CoSyNE on the double pole balancing task. However, benchmarks have been criticised by Stanley stating

"The problem is that benchmark performance may not correlate to the long-term goal of the field, which is to discover encoding with the expressive capacity demonstrated by the complexity of natural organisms" [214].

With this in mind, NEAT and other TWEANN methods provide the unique trait of further abstracting the human experimenter from the task in the aspiration that evolution alone is capable of finding an appropriate solution. HyperNEAT has proven itself in a great number of varied domains, including the desirable trait of scaling up to domains which would require workarounds with NEAT, such as in Go. Both share many similarities in terms of the evolutionary process due to NEAT being used in the construction of the CPPN. The connection between the two allows changes in the task domain or evolutionary process to have reasonably similar effects. Both, also, have a wide array of valuable extensions which provide avenues of further exploration.

Genetic diversity shows a great variety of ways in which evolution's outcomes can be enhanced with simple changes to the evolutionary process; with many compatible with each other. However, each will not necessarily be a direct and effortless solution to a problem. Take for instance NS, this would not be effective in an environment that already provides a clear gradient to the result. Encoding techniques may also include diversity mechanisms as part of their established benefit; such as concepts outlined in crowding and fitness sharing are seen in speciation within ESP, NEAT, HyperNEAT etc. The section provides resourceful methods and means to overcome common problems in NE if they become evident during experimentation; but, at this stage would disturb results which could already be achieved with a simple GA.

3| Experiment Domains

3.1 Overview

Simulations in evolutionary computing allow researchers to evolve and study behaviours through observable periods of evolution in an achievable and realistic time frame. These types of simulations can range from coevolution [267, 27], morphological evolution [8, 18], predator and prey behaviours [71] etc. Computer models may not represent the biological reality with sufficient fidelity and it is unclear whether conclusions drawn, in silico, can be transferred to the carbon-based biological medium; yet, evolutionary computing can still provide an understanding of how things work [58]. In the case of this thesis, the predominant focus is creating an evolvable motion planning network that reproduces the behaviours of artificial potential fields; used prominently in evolutionary robotics [247, 240]. Motion planning may appear as an engineering challenge, however, when combined with other networks, motion planning can produce agents with life-like responses in a physics-based environment; a natural biological phenomena to observe [222]. A simplified version of this domain will be used for this work known as the River Crossing Task (RC Task) (section 3.2). The RC Task has been used numerous times as a test domain and the discoveries from this research are instantly applicable to those works, and other domains like it. Later in the thesis, a common multi-domain benchmarking platform known as the Arcade Learning Environment (ALE) (section 3.3) is utilised. This provides a variety of domain types and is useful for evaluating the general-purpose competence of evolvable solutions; allowing researchers to compare and contrast performance.

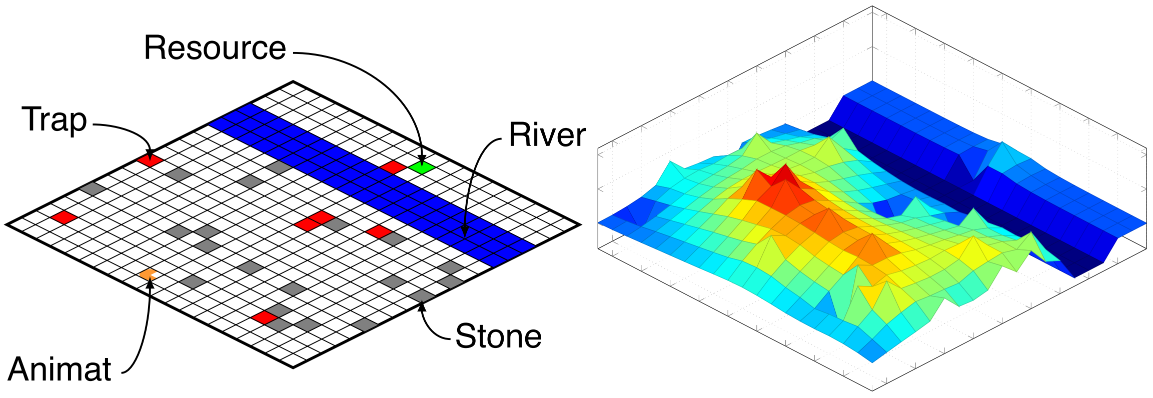


Figure 3.1: **RC World environment (left) with the corresponding activity landscape (right) produced via Shunting Model (SM).** Activity from desirable states propagates through the network to create an activity landscape. The dynamic activity landscape is used to determine where the next robot position is. Red is the highest activation value; Blue is the lowest. Clustered desirable objects cause greater areas of activity (see stones compared to resource).

3.2 River Crossing Task (RC Task)

The RC Task is a hierarchical task where higher-level decisions require the use of lower-level skills. It was devised in Robinson et al. [192] to demonstrate high-level deliberative and reactive behaviours produced by a modular neural architecture. The neural architecture consists of a Shunting Model (SM) (section 3.2.2) with static weights and a Decision Network (DN) (section 3.2.1) with evolvable weights. The DN's weights were evolved via a steady-state genetic algorithm. Fitness is based upon animat's ability to locate a resource in a 20x20 discrete world. Then, tournament selection was used for each iteration, with three animats evaluated and the worst performer replaced by a new offspring created from a combination of the other two. The goal of an agent is to navigate to the target resource in each two-dimensional RC world, within 100 time-steps, while avoiding harmful objects. Once the resource is obtained, the agent proceeds to the next RC World. Each world increases in complexity via an expanding river obstruction between agent and resource. Agents must evolve to move randomly positioned 'stones' to build bridges to cross the river, in order to complete the more challenging RC World environments.

RC worlds are constructed on a 20x20 bounded grid in which each cell contains zero or one of each of four object types: stone, trap, water, and resource. A cell containing none of these four object types is deemed to contain grass. Stones and traps are initially placed into each environment at random while water is positioned deterministically. Moving on to a trap or water kills an agent. Stones are movable objects: they can be picked up and dropped on grass or water. If one is dropped on water then the water object is converted into a grass object. Complexity is enforced by water placed across the world, creating a river obstacle between the agent's starting position and the resource. When placed on water, stones can be used to build a bridge. Fitness is a discrete value from 0 to 4, determined by the number of RC worlds in which the agent reaches the resource. Hence rewards are sparse, with no contribution to fitness from partly completing an environment by moving nearer to the resource or building part of a bridge. Agents are evaluated first in a world with river width 0 (no river), then 1, 2, and 3, stopping at first failure.

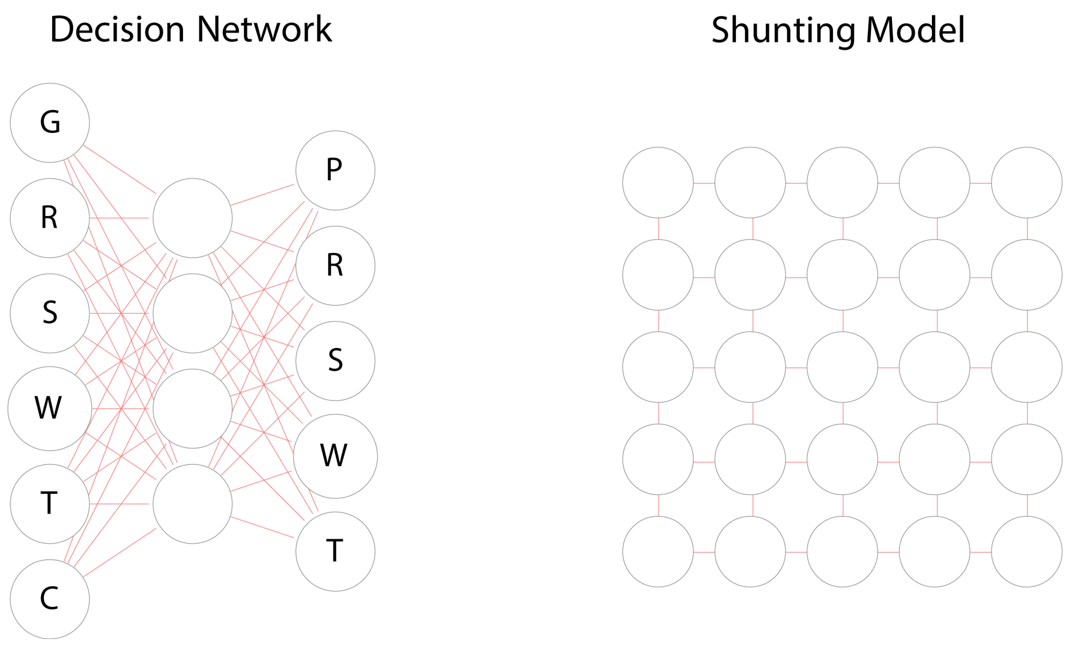


Figure 3.2: **Neural Architecture for the RC Task.** On the left, the decision network controller. The output neurons are P = pick up/put down; R = resource; S = stone; W = water; T = trap. The input neurons are G = grass; R = resource; S = stone; W = water; T = trap. On the right, the shunting model a motion planning model. Each neuron represents a possible state of the system and is connected to a subset of it's Moore neighbourhood; this subset is called the receptive field, and represents all the states that are reachable from the current state.

3.2.1 The Decision Network (DN)

The DN is a standard feed-forward neural network. Topology consists of six inputs, four hidden nodes and five outputs. Five inputs are binary to represent the presence or absence of each object type (including one for grass) on the current agent's position.

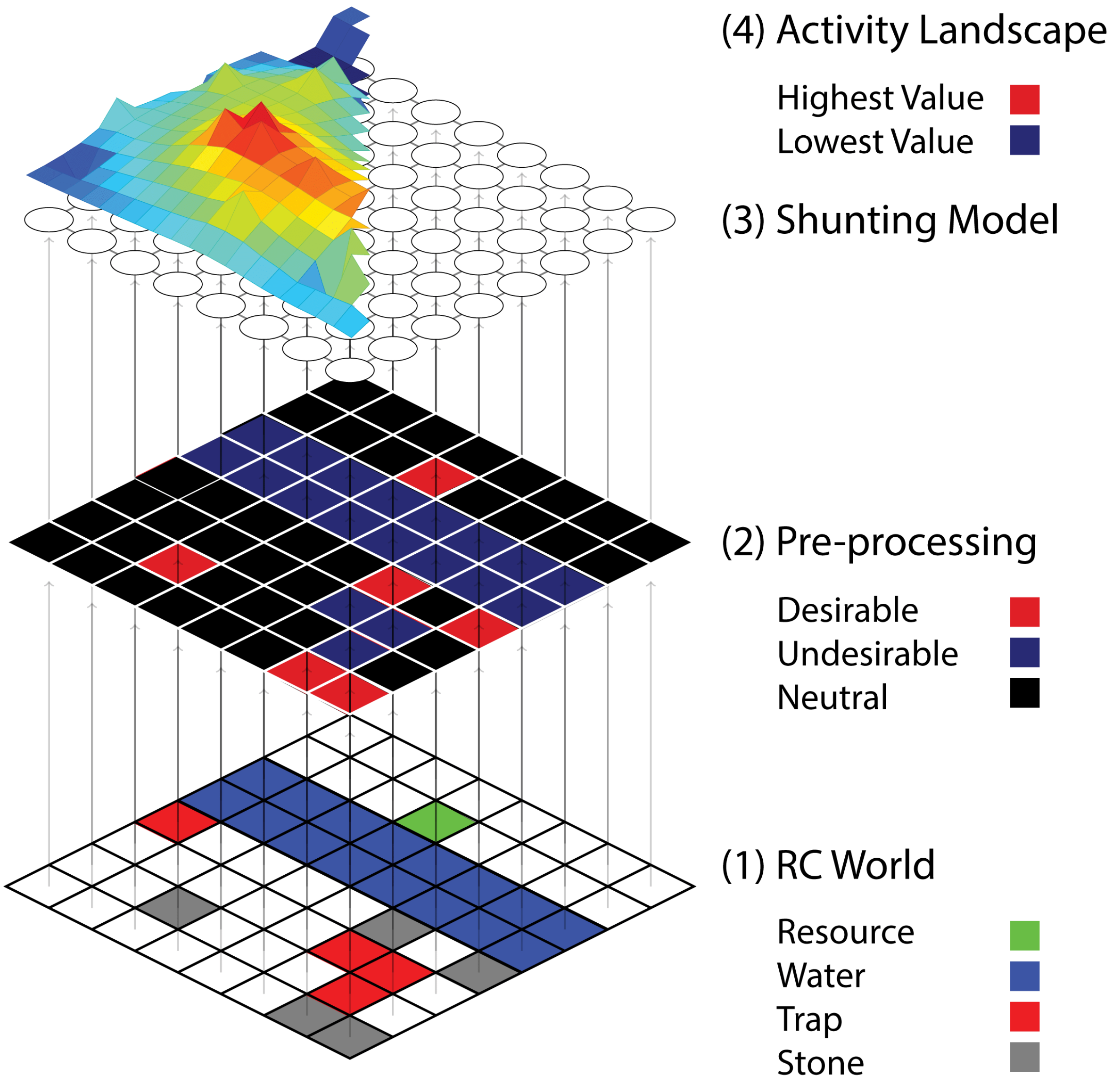


Figure 3.3: **Illustration of the relationship between the RC Task and neural network.** Attributes at the agent's position determine inputs to the Decision Network. Attributes of the RC World are converted to iota values via the Decision Network outputs and mapped to the corresponding position of the attributes (2). This is a static pre-processing stage to prepare the inputs for the Shunting Model. A 20x20 matrix of iotas are passed to the Shunting Model and activated (3). The activity landscape is a visual representation of Shunting Model's output after completion.

The sixth input represents whether or not the agent is carrying a stone. Four outputs correspond to the desirability of each object type and the fifth determines whether the agent should pick up (positive output) or put down (negative output) a stone; the architecture can be seen at figure 3.2. All hidden and output neurons use a hyperbolic tangent activation function. Output values that are below, within, and above the range $[-0.3, 0.3]$ are converted to -1, 0, and 1 respectively. The normalised output values from the DN are multiplied by 15 to supply numbers that indicate the desirability to the agent of objects in each cell of the environment. We refer to these desirabilities as *iota values*. Desirable objects refer to those that agents would like to traverse to, undesirable objects are one in which agents seek to avoid.

3.2.2 The Shunting Model (SM)

The SM is a topographically-ordered neural network; mapping the environment domain to discrete neuron positions in the network. The function is to produce a collision-free trajectory between two positions in a dynamic two-dimensional environment. The trajectory being a varying path that changes as the environment does. By following a steepest gradient ascent rule, agents can maneuver to desirable locations while avoiding undesirable objects. Each neuron in the neural network has only local connections to its Moore neighbourhood. The trajectory is generated without explicitly optimising any cost functions and without any learning processes.

First used in Meng and Yang [148], the SM was applied to real-time robotics to solve maze-type problems by mapping the physical environment to positional neurons.

Activity from desirable states propagates through the network to create an activity landscape. The dynamic activity landscape is used to determine where the next robot position is. Peaks form at objectives and troughs at undesirable positions. In the RC Task, the input for the SM are the objects *iota values* produced via the DN and placed in positions that topographically correlate to the RC world's object positions. The activity values are then produced by diffusing iota values via the following equation:

$$x_i^{new} = \min\left(\frac{1}{8}\sum_{j\in N_i}\left[x_j\right]^+ + I_i, \max I\right) \tag{3.1}$$

where x_i^{new} is the activation of neuron i; I_i is the external input determined by the iota value of the object present in cell i; N_i is the receptive field of i and $\max_i$ is the maximum iota value (15). The receptive field represents all the states that are reachable from the current state, in this case the cells in its Moore neighbourhood. Usually, the receptive field is eight cells but boundary cells vary from three to five cells. Equation 3.1 is iterated fifty times to allow activity to propagate and stabilise across the 20x20 array of SM neurons, as shown in figure 3.4; the amount of times equation 3.1 is iterated is dependent on the size of the RC World/configuration space. Agents move to the neighbouring cell with highest activation (gradient ascent). Early RC implementations of the SM used a real-time (differential) form of this update equation but in more recent work the implementation has changed for simplicity and clarity while maintaining the desired behaviour [222, 223]. This implementation keeps all weights static at $\frac{1}{8}$.

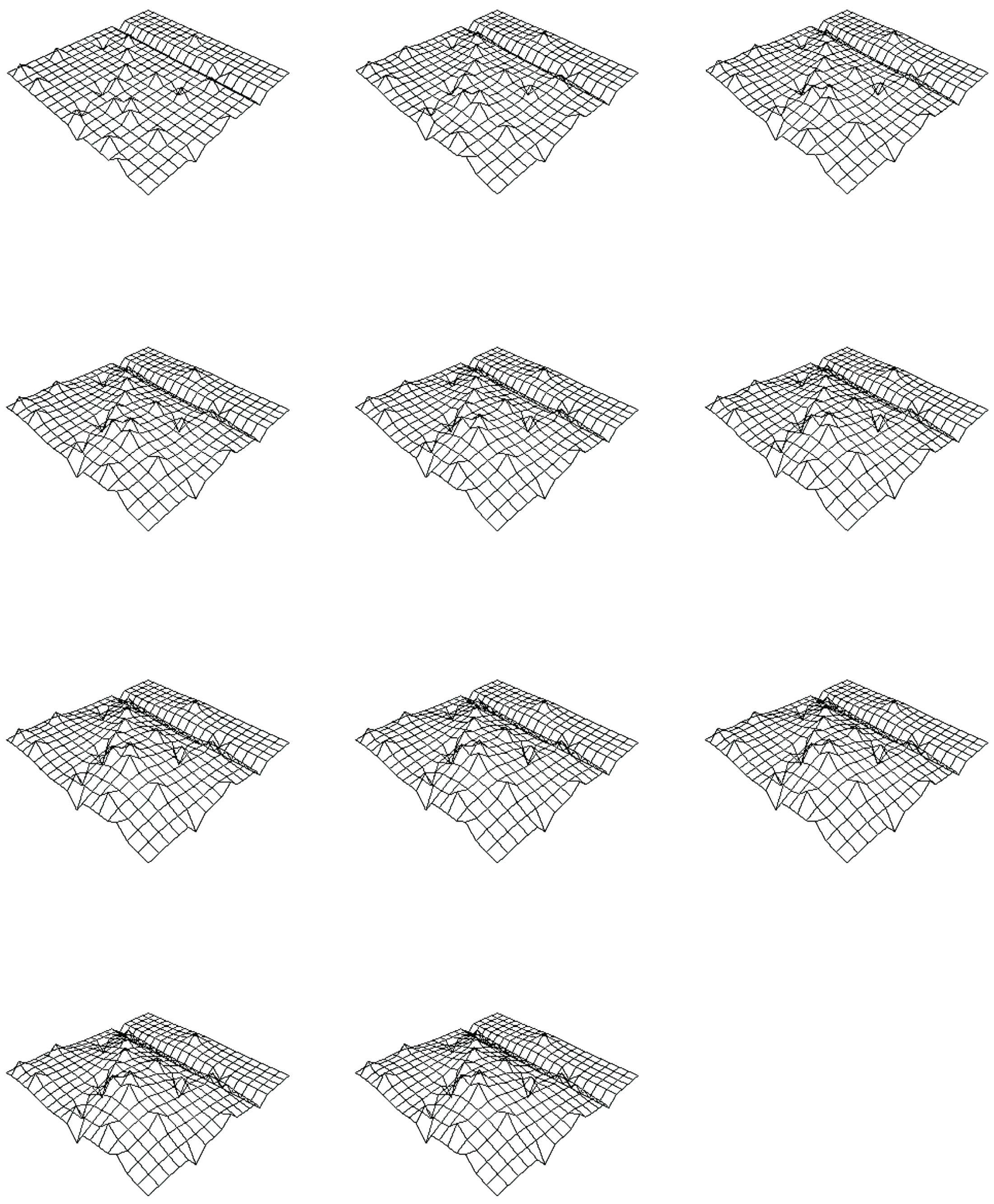


Figure 3.4: **Visual of activity landscape from iteration 1 to 50**. From left to right, vertically down, a snapshot is taken of the activity landscape at various iterations of equation 3.1. Activity from desirable states propagates through the network to create an activity landscape. Peaks form at objectives and troughs at undesirable positions.

3.3 Arcade Learning Environment (ALE)

ALE is an evaluation platform that poses the challenge of building AI agents with general-purpose competency across Atari 2600 games. Originally introduced by Bellemare et al. [13], ALE provides a standard interface in which agents provided with raw pixel information choose an allowed action. Agents are provided no game-specific information. The challenge is to find policies through a high-dimensional state representation on a variety of tasks; evaluating the general-purpose competence of an agent. As each game requires a different strategy, agents are able to be compared and contrasted in where they excel or exhibit a weakness. For example, APE-X vastly outperforms a majority of any counterpart strategies [99]. But, DQN [157], A3C [158] Deep Neuroevolution [226], HyperNEAT [94], Evolutionary strategies [194] etc clash from game to game for superior results. Atari games are free from experimenter's bias, in which a task has been constructed to demonstrate a specific behaviour. Instead, the creation was intended as a human challenge, which in turn provides an interesting comparison to human performance.

HyperNEAT was one of the first major successes in utilising the ALE platform. This was later supplanted by the highly publicised DQN. DQN was the first algorithm to achieve human-level control in a large fraction of Atari 2600 games; this architecture can be seen in figure 3.5. Beyond this point, research remained largely with reinforcement algorithms with a broad array of techniques, such as Gorila DQN [170], DDQN [243], Prioritised DDQN [198], Dueling DDQN [249], A3C [158], NoisyNet [63], Distributional DQN [14], QR-DQN [40], Rainbow [98], APE-X [99], IQN [41] etc. But, with the introduction of deep neuroevolution and evolutionary strategies there was a renewed interest in pursuing evolutionary techniques with large ConvNets. Especially considering deep neuroevolution was achieved using a simple GA and not implementing the breadth of performance improving techniques developed over the decades in low dimensional NE domains, see chapter 2.

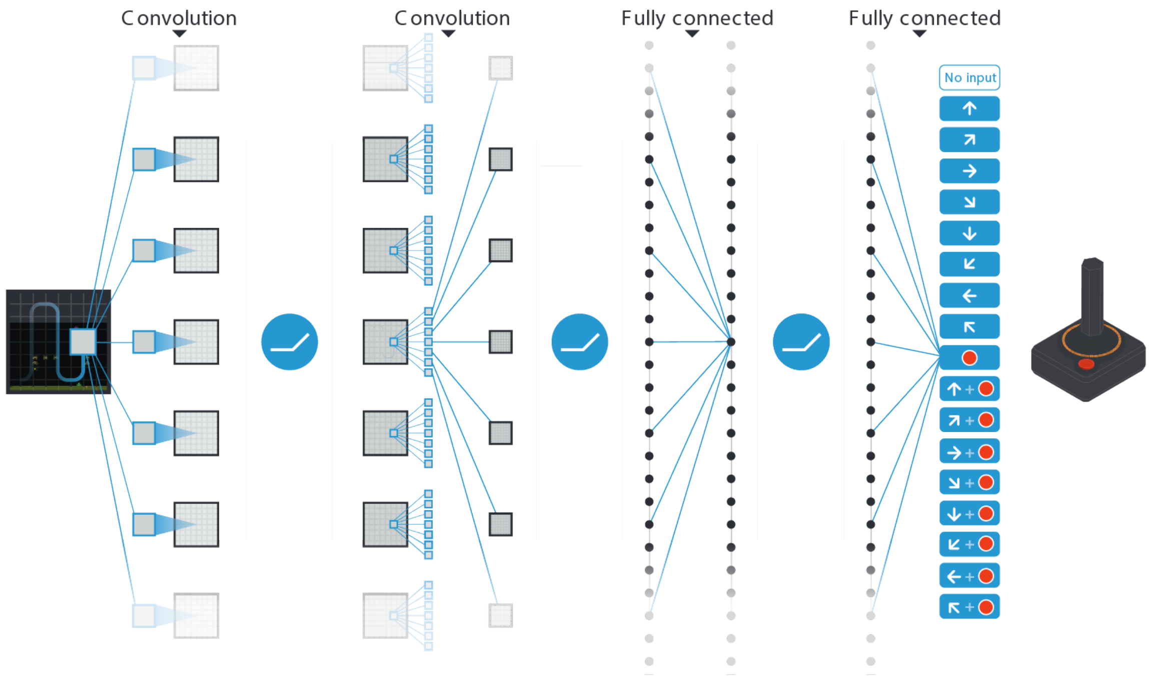


Figure 3.5: **Schematic illustration of the convolutional neural network for DQN in ALE.** The input to the neural network consists of an 84x84x4 image produced by the pre-processing map w, followed by three convolutional layers (note: snaking blue line symbolises sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier non linearity (that is, $max(0, x)$).

4 | Evaluating Scalable Motion Planning with Direct and Indirect Encoding

4.1 Introduction

This chapter examines the practicality of using NEAT and HyperNEAT as a solution for motion planning at scalable environment sizes; in this specific instance, the replication of SM behaviour (section 3.2.2). The RC Task has shown that when a hybrid neural architecture is tasked with completing a sparse reward challenge, it is able to do so while exhibiting difficult reactive and deliberate behaviours. This hybrid model relies heavily on its motion planning system (the SM) with its behaviours produced via a static equation. Therefore, the motion planning system is unaffected by an evolutionary process and ultimately restrains the model to this specific domain, or tasks similar to it. Yet, the behaviours produced via this model are desirable for other architectures, and domains which require greater long term planning. The goal of this chapter is to replace the static motion planning network with an evolvable solution (NEAT and HyperNEAT) that can change the path planning behaviours to the domain. This will be split into two sections, first scalability.

Motion planning approaches like the SM (artificial potential fields) convert the task environment, or physical space, into a configuration space, a finite virtual space. This requires the utilisation of the entire environment for paths to be created between any two points. For the RC Task, this will mean that the RC world will be the configuration space and will be the input for the evolvable solutions. The output is the same dimensionality as the input and creates a two dimensional space of gradients, with peaks and valleys; this is referred to as the activity landscape. As a result, the size of the RC world will be a contributing factor in the difficulty of this task; potentially larger genomes and a need for greater deliberate path planning. Scalability focuses on whether evolvable solutions can utilise an increasing input size to produce adequate path planning to complete the task. In the original RC task, the world was a 20x20 discrete matrix, but before attempting this large input size, the task domain will be reduced in complexity. The simplest possible RC world instance will be evaluated and incrementally increased; all while monitoring agent's performance. This will assess whether our chosen methods (NEAT, HyperNEAT) are capable at scaling.

Next, the quality of the solutions will be an important consideration. If evolvable approaches are not meeting similar, or equal, standards compared to the static approach they would not be adequate as replacements. The factors that will be considered are efficiency, training, and robustness. Training will be a comparison between approaches as the SM does not undergo training, but both efficiency and robustness can be compared to the static solution. Efficiency will express to what quality were the motion planning solutions producing the shortest possible paths to complete the task in the same environment. For a successful RC Task solution, multiple different paths are needed at various stages, especially during the bridge creation stage. Robustness will express how successful each motion planning solution is on a variety of different RC worlds. For an evolvable solution to be applicable both of these will be important traits.

4.2 Experiment #1

In this initial work, NEAT and HyperNEAT will participate in an evolutionary process to achieve motion planning behaviour that could be effective in a task that requires long term deliberative planning (RC Task). The focus of these experiments is to assess if either could scale a configuration space (input) to the original RC world size (20x20) while still being effective at motion planning. NEAT and HyperNEAT evolve networks that receive input as a representation of the current world state and output an activity landscape; an identically sized space with gradients of peaks and valleys. The RC Task is executed as laid out in section 3.2 and follows a gradient ascent rule on the network's output. Restrictions on the RC Task are laid out below; including how the task world is able to scale and exclusion of the DN in the evolutionary process. To be successful, networks must be able to create motion planning rules that complete the RC world at its highest difficulty, in this case, building a connecting river of three stones and reaching the resource. The evolutionary process of NEAT and HyperNEAT remain the same as laid out in sections 2.1.5 and 2.2.6, respectively. Although, both have their own independent parameters which are explained in the sections below and justifications for the parameters chosen for this experiment setup. Experiments have 10 evolutionary runs each for 200 generations, or until the highest fitness is achieved (4.0).

4.2.1 Experiment Setup

RC Task

The focus of this experiment is replacing the SM from the RC Task; that is, a network able to create pathways to desirable objects in a stochastic environment. For this to be the focus, aspects of the RC Task are restricted. First, the DN will not be included in the evolutionary process, instead, it will evolved to the fittest behaviour with the original SM and remain at that state while NEAT and HyperNEAT evolve; this is referred to as the pre-evolved DN. This is necessary as the DN dictates what is desirable and undesirable in the environment. The correct iota values are necessary for a motion planning network to complete the task domain. If they were to evolve concurrently then focus is taken away from strictly motion planning, which is why only the motion planning network (NEAT and HyperNEAT) will be evolved.

Next, the SM has the ability of obstacle avoidance but this is not a feature of this network, instead, the focus is purely on the rudimentary behaviour of creating pathways to desirable objects. As opposed to completely removing traps from the environment they are mitigated. This has been decided for two reasons. First, the rudimentary behaviour of deliberate traversal to activity peaks is not yet known and should be the focus before adding to task complexity. Secondly, both NEAT and HyperNEAT would need further network considerations, such as topology or activation function changes. (1) Activation functions would have to change to ReLU, which is not a common function with standard NEAT and HyperNEAT approaches and (2) HyperNEAT would require an addition of a hidden layer or separate inputs for negative and positive values. This appears to be future consideration and out of the scope for this chapter. To mitigate traps, a negative iota (-1) value will be applied to the activity landscape (the output of NEAT and HyperNEAT) at the position of traps.

RC world size is the next contributing factor to performance. Each input of both NEAT and HyperNEAT will be the configuration space of the RC world; configuration space being the physical space translated to a finite virtual space. Each input cell will map to discrete positions in the RC World. So, inputs are represented as two-dimensional planes of two discrete integers (4x4,5x5,6x6 etc). The values passed to the networks will be the iota values of each object in the environment, leaving a two-dimensional matrix of 1s, 0s, and -1s. Direct encoding will see a square growth in the initial genome size as the RC world grows. This is due to the genome containing each weight for the network and connection size will grow by the square of input, this is clearly illustrated in figure 4.2. Whereas, the genome in the indirect encoding method will not scale; only the substrate size will require changing. To target this for analysis the world will be incrementally scaled. Starting at the smallest possible size (4x4) the world will increase towards the original size (20x20); Traps and Stones scale also.

Agents will follow the highest activation neighbour in their Moore neighbourhood with radius 1. This remains the same from the conventional RC Task and other HyperNEAT implementations [93].

Fitness Function

The original fitness function used by Robinson et al. [192] (i.e. the number of RC worlds in which the agent reaches the resource) is not a good assessment of the agent's general performance. As this work aims to find an SM replacement, general performance is a core aim. Rather than the original fitness function (the number of RC worlds solved) for greater granular feedback, the new fitness function uses the mean performance score from 10 RC Task runs. Agents will have to consistently build bridges in different environments to survive. From a biological perspective, the task is now taking place in a larger RC environment; with the 10 RC worlds being interactions within the larger world. Also, greater fitness granularity allows the evolutionary process to exploit speciation; which both approaches use. By allowing 41 discrete fitness values, as opposed to 5, individuals are able to form in species (section 2.1.5) which represent their general performance and allow greater diversity in the population.

NEAT

Stanley et al. (2002) established the importance of an initial population of small fully connected networks using NEAT. However, as the RC world sizes grow, there will

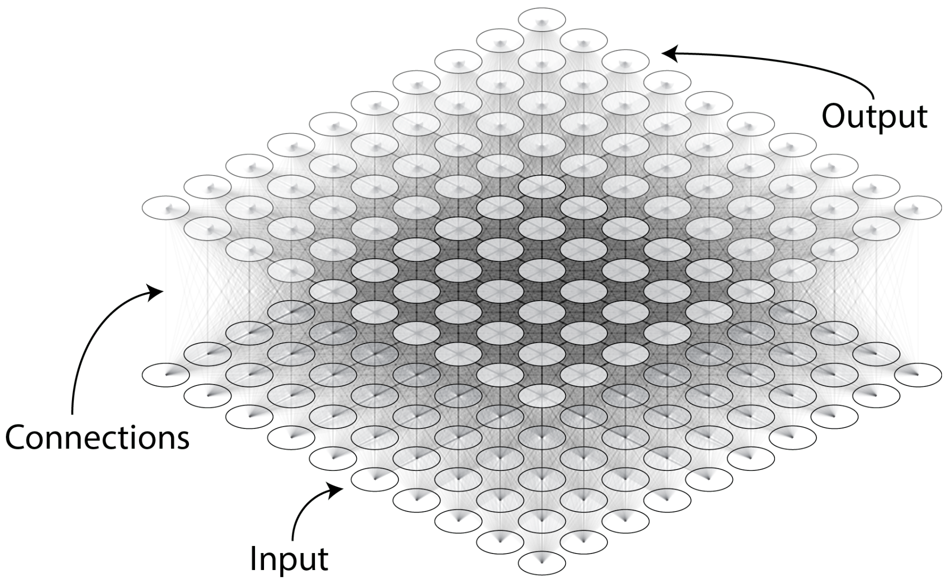


Figure 4.1: **Illustration of the sandwich substrate.** A single two-dimensional sheet of neurons fully-connected to another two-dimensional sheet. Used as a common HyperNEAT substrate.

be no apparent way to keep network size and connections size down. Other works explored networks with no initial links, requiring mutations to form the network [252]. However, as the RC world increases, a non-connected network would require greater link mutations in order for every cell to comprehensively communicate. So, for this model, the initial population will continue to be fully connected. In some ways, this approach is closer to a static ANN and does not exploit the properties of TWEANN. However, NEAT has shown an ability to perform as a static ANN solution over standard direct encoding methods. The topology will consist of a fully connected single layer, including recursive links depending on the parameters chosen.

NEAT, and HyperNEAT, has over 30 parameters to adjust the evolutionary process. These parameters encompass; all possible mutations; various crossover approaches and speciation options, such as threshold percentages. To rule out the possibility that NEAT's parameters are tuned incorrectly for this experiment, a broad set of four NEAT parameters sets were chosen across the NEAT package suite, which have shown success in other challenging tasks. Each includes unique traits. Around 60% of the parameters in each set are equal, notably producing identical parameters for crossover, though the sets differ in their approach to topology mutation, weight mutation, speciation, and population size. Below are the key differences between each strategy:

- **NEAT01** - Small population. A focus on weight optimisation rather than topology.
- **NEAT02** - Large population with large number of species.
- **NEAT03** - Large population with small number of species.
- **NEAT04** - Small population. Small Mutation Rates. Significantly smaller drop-off age than NEAT01.

Above highlights the main comparative differences between the data sets. The full list of parameters differences can be seen in table 4.1 with an explanation of the parameter's function.

HyperNEAT

HyperNEAT uses two networks, a CPPN and a substrate. The CPPN encodes the connectivity pattern of the substrate. The substrate is an ANN whose nodes simulate

NEAT Parameters	**01**	**02**	**03**	**04**	**Explanation**
weigh_power	2.5	1.8	1	1	Power of a link weight mutation
mutdiff_ coeff	2	3	7	2	Coefficient for mutation differences for speciation
compat_ thresh	3	4	9	3	Threshold for the genotypes the same species based on the compatibility number
survival _thresh	.2	.4	.4	.2	Percentage of a species population that reproduce
mutate_link_weights	.9	.8	.8	.8	Probability to mutate link weights
mutate_toggle_enable	0	.1	0	0	Probability to toggle genes on or off
mutate_gene_reenable	0	.05	0	0	Probability to find the first disabled gene and enable it
mutate_add_node	.0025	.01	.01	.0025	Probability to mutate genome by adding a node
mutate_add_link	.1	.3	.3	.1	Probability to mutate the genome by adding a new link between two random Nodes
recur	.2	.05	.05	.05	Convert a link to a recurrent link
interspecies_mate_rate	.05	.001	.01	.001	Probability to mate outside of species, with a bias towards better species
pop_size	256	1000	1000	256	Population Size
dropoff_age	1000	15	15	15	Dropoff age dictates when stagnation has taken place in the population by comparing the number of generations since the previous fittest individual. If beyond this parameter species are consisered dying and offspring will not be given to that species.

Table 4.1: **Parameter differences for each NEAT experiment set.** Each parameter is explained on the right hand column.

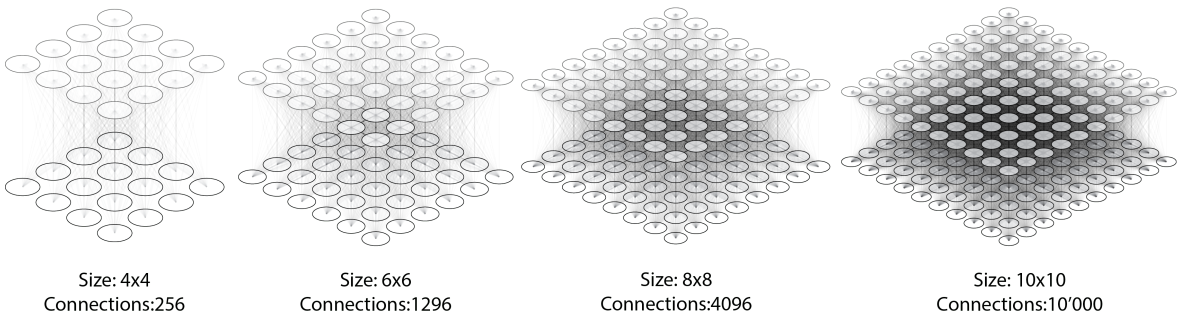


Figure 4.2: **Visual example NEAT & HyperNEAT’s topology for various RC world sizes. NEAT’s topology may change throughout the evolutionary process.**

a coordinate system to represent the topology. The substrate design is important to the performance of HyperNEAT, but general-purpose substrate designs have proven successful in a variety of different task domains [32]. The *sandwich substrate* is one highlighted in existing literature; this is a single two dimensional sheet of neurons fully-connected to another two-dimensional sheet; can be seen at figure 4.1. It has been used successfully in locomotion tasks in which there is no obvious geometric relationship between sensor positions and substrate inputs [32, 130]. To avoid the bias of expert domain knowledge, this work uses the sandwich substrate with 20x20 input and output layers. Additionally, links with weight values less than 0.2 or greater than -0.2 are retained to allow greater nuances in the encoding pattern; previously these were discarded.

HyperNEAT’s parameters have been found vital for tuning [213]; however, the parameters within the original package for Checkers, Go, and Robot Arm remain similar with only the topological mutation rate and population size deviating for each experi-

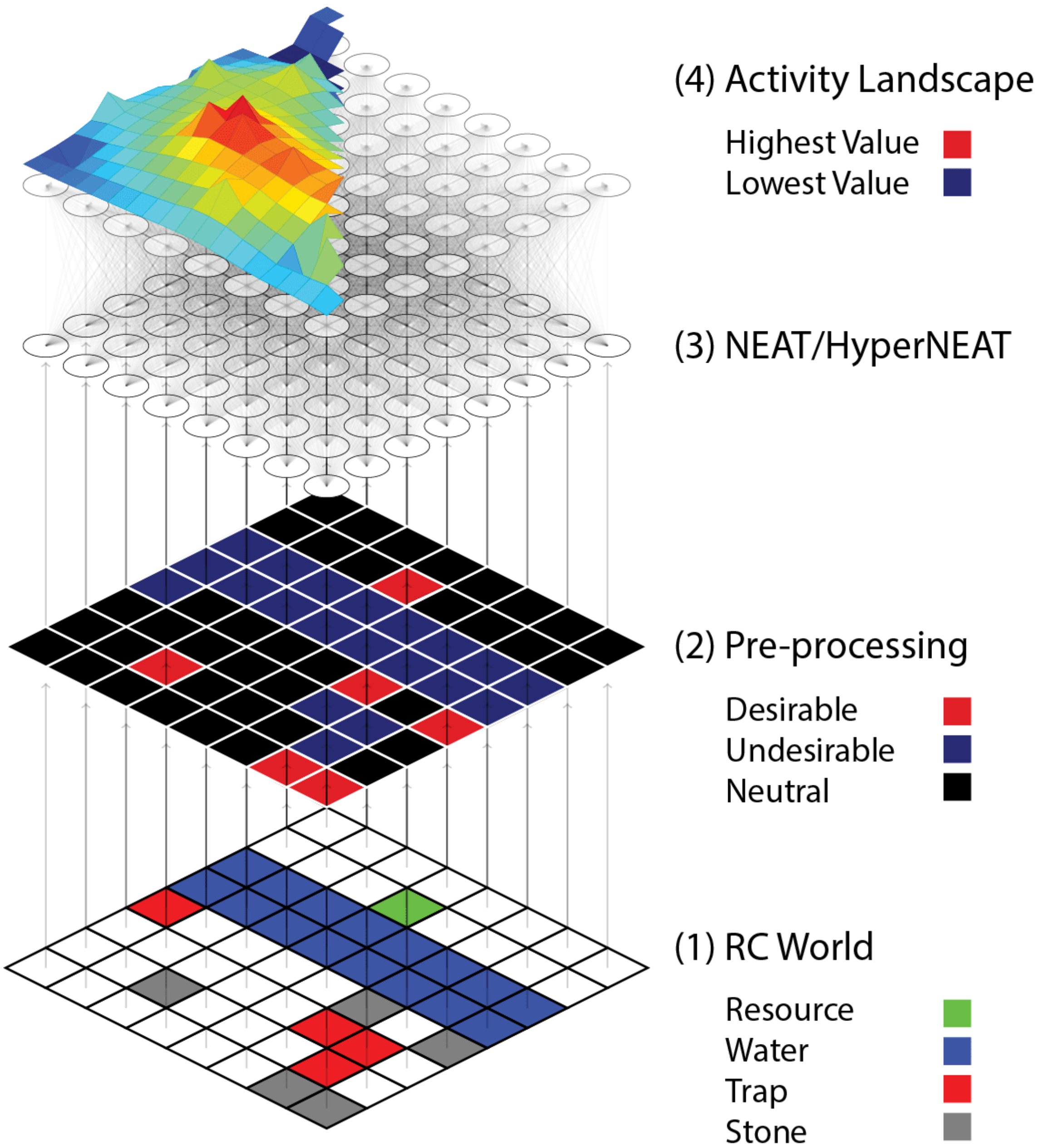


Figure 4.3: **Neural architecture for experiment #1 & #2.** Attributes at the agent's position determine inputs to the Decision Network. Attributes of the RC World are converted to iota values via the Decision Network outputs and mapped to the corresponding position of the attributes (2). This is a static pre-processing stage to prepare the inputs for the NEAT/HyperNEAT. A 20x20 matrix of iotas are passed to the NEAT/HyperNEAT and activated (3). The activity landscape is a visual representation of NEAT/HyperNEAT's output after completion.

ment. These experiments are distinctively different in what they are trying to achieve in their task domain. The robot arm experiment models the kinematics and dynamics of a two-dimensional muscular hydrostat as a chain of quadrilateral polygons with fixed area connected to a fixed base [257]. Inputs are provided via sensors that infer the position of each segment relative to a target. The goal is to reduce the distance to the target; with training reaching the smallest distance within 3000 generations and 16 arm segments. Checkers [66] and Go [67] both share an identical substrate, yet are both able to adapt to each domains rules. Go, winning 8 out of 10 games on a $5x5$ board, and 6 out of 10 on $7x7$ after 100 generations. And Checkers, achieving an average of 60% of wins on a $8x8$ board. Each experiment has found success in their domains and therefore, without specialist knowledge on tuning parameters, the default parameters should offer no resistance in HyperNEAT achieving a general close representation of what it can achieve overall. As tasks become more complex, the original packages seem to favour a lower than average topological mutation rate (0.05%) and a large population size (1000); these changes will be used in this work.

Finally, HyperNEAT is used in this work over the more advanced HybrID and ES-HyperNEAT, due to this task's requirements. Until recent extensions to HybrID, which uses both direct and indirect encoding, a pre-known evolutionary generation was required to switch encoding strategy once performance had plateaued [97]; domain specific knowledge, such as this, will be avoided in this work. ES-HyperNEAT is only applicable for networks with hidden layers, which this task does not require, due to the theoretical simplicity of a successful encoding pattern.

Table 4.2: Possible activation functions

Type	**Equation**
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Sinusoid	$f(x) = \sin(x)$
Gaussian	$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$ a, b, and c are constants
Identity	$f(x) = (x)$

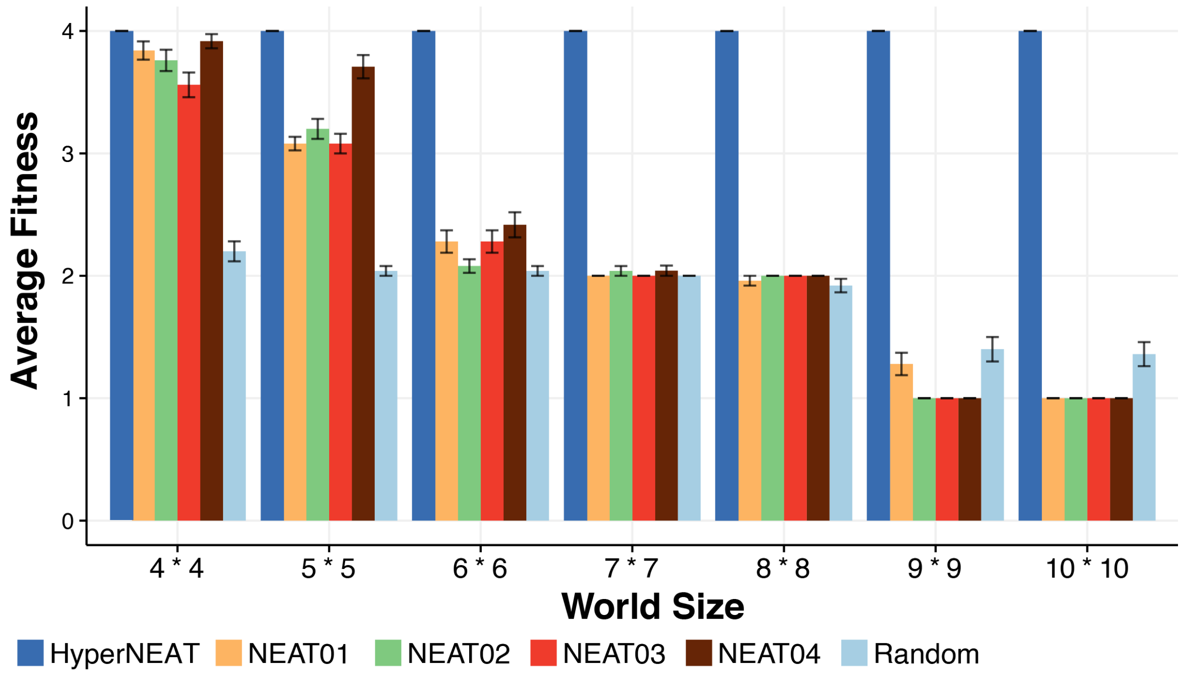


Figure 4.4: **NEAT, HyperNEAT, and Random mean fitness on various RC world sizes.** Fitness is taken from fittest individuals from twenty-five evolutionary runs of the RC Task. Error bars represent standard deviations. Fitness is the mean score over 10 RC Task worlds.

4.2.2 Results

Figure 4.4 compares the fittest individuals of all approaches obtained from twenty-five evolutionary runs. The scores show the mean fitness of the fittest individuals in each run. The results shown are taken from RC world 4x4 to 10x10. A negative correlation can be seen in all NEAT approaches as the world size increases. Using a Two-Sample t-test ($P < 0.001$) between NEAT and Random, all NEAT parameters achieve a statistically significant difference in smaller world sizes (4x4 and 5x5). However, as the world size increases, each NEAT parameter set becomes indistinguishable from Random. This persists until 10x10 in which Random then takes a statistical advantage. In contrast, HyperNEAT was able to consistently find a behaviour that could achieve the highest fitness in every run on each world size up to 20x20, not displayed in Figure 4.4.

To analyse the most successful networks further, training continued on 4x4 until there were 10 successfully trained networks that were able to complete RC world at river width 3; the results can be seen in table 4.3. The features of the evolutionary process are captured as mean averages and all sets of 10 runs undergo a two sample t-test. When referring to significance the p value between sets were below 0.05. Observing the data in table 4.3 and the parameter differences at table 4.1, *'dropoff_age'* and *'pop_size'* appear to have the most impact on how evolutionary features appear. First, the focus will be on topological growth, which is additional topology features added after the initial genome (node and links).

Each parameter set saw no significant difference in additional links created; all averaging around 26-38 per successful solution. It might be expected for NEAT02 and NEAT03 to produce a greater amount of links due to the higher *'mutate_add_link'* variable. The lack of significant difference is likely tied to the statistical similarity of the

	Mean Average			
NEAT	**1**	**2**	**3**	**4**
Node	1	2.4	3.3	1.6
Links	30.9	30.7	37.9	26.2
Age	335.7	7.1	9.2	12
Generations	474.1	93.7	129	118.9
Species Num	180.8	793.3	523.1	243.1

	P Value			
	Node			
	1	**2**	**3**	4
1		0.031429	0.000954	0.000954
2	0.031429		0.286801	0.250205
3	0.000954	0.286801		0.013225
4	0.000954	0.250205	0.013225	

	Links			
	1	**2**	**3**	**4**
1		0.972788	0.207682	0.347504
2	0.972788		0.332344	0.521124
3	0.207682	0.332344		0.089649
4	0.347504	0.521124	0.089649	

	Age			
	1	**2**	**3**	**4**
1		**<.00001**	**<.00001**	**<.00001**
2	**<.00001**		0.433538	0.099622
3	**<.00001**	0.433538		0.291547
4	**<.00001**	0.099622	0.291547	

	Generations			
	1	2	3	4
1		**<.00001**	**<.00001**	**<.00001**
2	**<.00001**		0.117403	0.298518
3	**<.00001**	0.117403		0.660978
4	**<.00001**	0.298518	0.660978	

	Species			
	1	**2**	**3**	**4**
1		**<.00001**	**<.00001**	**0.041188**
2	**<.00001**		**0.003748**	**<.00001**
3	**<.00001**	**0.003748**		**0.000449**
4	**0.041188**	**<.00001**	**0.000449**	

Table 4.3: Table consists of 10 RC world runs at 4x4 size with each NEAT parameter set. The first table shows the mean of successful network solutions and their evolutionary features. This consists of; the number of additional nodes added; additional links added; the age of the species the network was within; a number of generations it took to obtain a solution and the number of species recorded through the evolutionary process. The remaining tables show *p*-values from comparisons of each parameter set's evolutionary features. Those in bold are statistically significant (p <0.05).

number of generations between NEAT02, NEAT03, NEAT03 not allowing enough evolutionary time to see divergent topologies. This, however, does not apply to NEAT01 which saw a statistically higher number of generations to find successful solutions. This is linked to NEAT01s much higher parameter for *'dropoff_age'*; which dictates when stagnation has taken place in the population by comparing the number of generations since the previous fittest individual. As a result, NEAT01's initial species are extremely improbable to die out due to the high drop off age being 1000 and generations to find a successful solution averaging around 474.1. Coupled with a much higher likelihood to have an inter-species crossover (*interspecies_mate_rate*) topologies similar to the original will propagate throughout the evolutionary process. NEAT01 leaning towards topologies with fewer nodes can also be explained by the bias towards weight optimisation in the parameter set; with the highest parameter value for *'mutate_link_weights'* and having the lowest for parameters which would cause topological mutation; *'mutate_add_node'*, *'mutate_add_link'* and *'recur'*.

The number of species generated sums the amount of new species created through the evolutionary process. Counting species is a loose metric of how diverse the population was through evolutionary training; the metric is loose as each set has different parameters for *'compat_thresh'* and *'mutdiff_coef'* which would affect how each set defines a species. Each parameter set saw a statistical difference to species generated compared to each other. A factor to take into account, with repeated runs of the same parameter, a correlation emerges between greater generations and greater numbers of species generated. So, comparing parameter sets is not a direct relationship, however, we can find some interesting observations. NEAT01 has a significantly lower number of new species despite having a significantly higher average generation count compared to other approaches. This again is due to the higher *'dropoff_age'* and reinforces what was stated earlier about a lack of diversity in the evolutionary process. NEAT02 and NEAT03 see a much higher sum of species due to the greater population of nearly four times that of NEAT01, NEAT04. The larger 'compat_thresh' in NEAT03 seems to explain the difference between the two sets as this would allow much larger differences in genotypes being part of the same species. NEAT04 displays, again, how a smaller population size creates less opportunities for diversity in the population. Yet, NEAT04 produced the best results of the parameters sets for worlds 4x4, 5x5, and 6x6. For this particular experiment, a reduced age drop off and reduced population appear to be the apt approach. This can only be deduced for smaller RC world sizes and beyond that direct encoding does not seem appropriate for this task.

4.2.3 Discussion

This chapter found that NEAT was unable to produce the fittest behaviour in the RC task consistently, even at the smallest world size. As world sizes increased the fitness decreased until agents could not make it past the first RC world. On the other hand, HyperNEAT was successful for all RC sizes and did so consistently. The scalability of indirect encoding is a known benefit but due to the success of smaller world sizes also suggests that HyperNEAT inherently has greater beneficial attributes for motion planning. This will be analysed in greater detail in the next chapter.

Scalability is already a known issue with fully connected direct encoding methods. There could be a suggestion that the incorrect architecture was used here. For example, with the game Go, NEAT used a roving eye technique which would limit inputs to a specific section instead of the entire board [218]. However, something like a roving eye creates behaviours that would focus only on the immediate neighbourhood. Although this may be a flexible solution for the agent to show reactive behaviour (i.e moving away from traps), agents wouldn't be aware of objects at a distance (i.e. the resource). Furthermore, NEAT was unable to produce appropriate motion planning at relatively small world sizes of 12 maximum cell positions, where HyperNEAT was successful. So rather than just scalability, NEAT's problem with motion planning may be due to the lack of weight patterns that exploit regularity and repetition. It could also be attributed that mutations would take generations to exhibit meaningful change, as it would be based on a node by node basis. A change in the CPPN in HyperNEAT allows a dramatically different weight matrix on the substrate via single mutation.

4.3 Experiment #2

The following experiment will be similar to the previous section (section 4.2) with four key changes:

1. The size of the RC world will remain at the original 20x20 as in Robinson et al..
2. The original and updated fitness function (section 4.2) will be evaluated together. Originally being the fitness over 4 RC Task worlds, the fitness modification being

the mean of 40 RC worlds.

3. Agents that achieve the highest fitness will be simulated through a Robustness Test; which simulates agents through 10^4 static RC world configurations with all river width sizes. This task assesses the agent's general performance.

4. Two CPPN input variations will be simulated. Standard CPPN inputs for the majority of HyperNEAT experiments include $x1$, $x2$, $y1$, $y2$, and a bias. The x and y positions are referencing the position in a 2-D plane of the input neuron and output neuron in the substrate. The input for these neurons are the normalised value of each neuron's position in the substrate. Delta inputs ($x1$ - $x2$) ($y1$ - $y2$) are included in certain experiments which allow patterns to emerge from relative distances. Delta may aid in the SM's reliance on relative patterns, as activity values degrade greater the further away from a desirable object they are.

   A comparison between delta and non-delta will reveal whether distance information yields benefits in this task, or if absolute inputs can achieve the same quality results, as seen in other work [65].

4.3.1 Results

For each HyperNEAT approach, twenty-five evolutionary runs for 200 generations were carried out on the RC Task. Each approach is separated with the use of a fitness modifier and delta inputs. Each method is named as follows:

- **F-D**: Fitness Modification with Delta inputs

- **F-*n*D**: Fitness Modification without Delta inputs
- ***n*F-D**: No Fitness Modification with Delta inputs
- ***n*F-*n*D**: No Fitness Modification without Delta inputs

The goal is to assess performance of each approach comparatively and against the SM with a focus on general-purpose performance. Figure 4.6 demonstrates this with an overview of a population's performance over generations using the original RC fitness function and the fitness modification (section 4.2).

Scores for the fittest individuals from all the evolutionary runs were collected and aggregated, as shown in figure 4.5. From this graph every run was able to solve the highest level of difficulty the RC Task requires, and maintain it. Each delta counterpart saw some statistical advantage in early generations over the non-delta inputs ($p <$

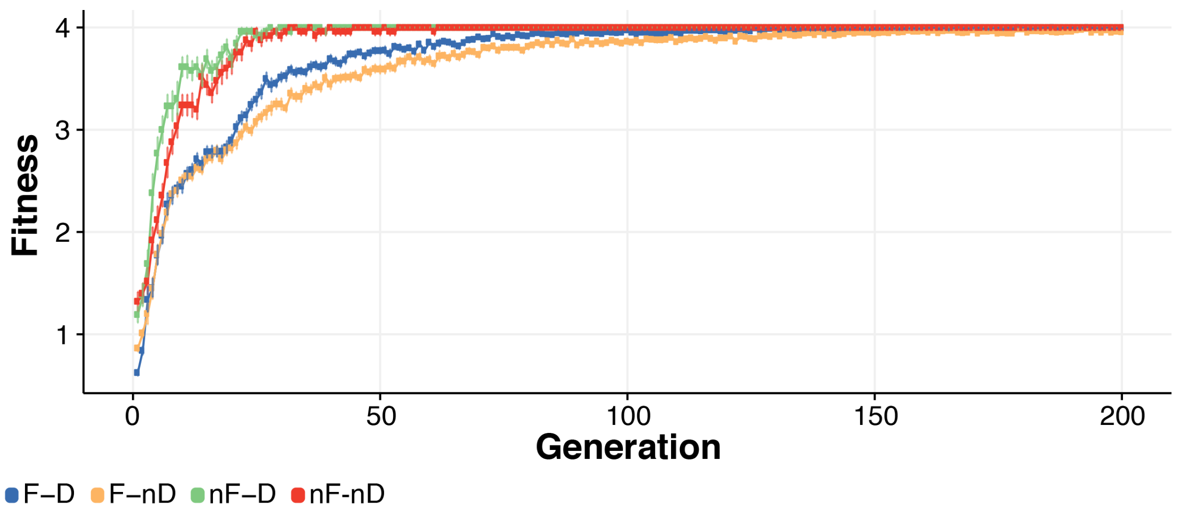


Figure 4.5: **Mean fitness of the fittest individuals from twenty-five evolutionary runs of the RC Task.** Error bars can be seen at each point of fitness and represents standard deviations; some error bars are obscured by the data points.

0.05), but this fluctuated each generation and by the 123rd generation there was no statistically significant difference between strategies.

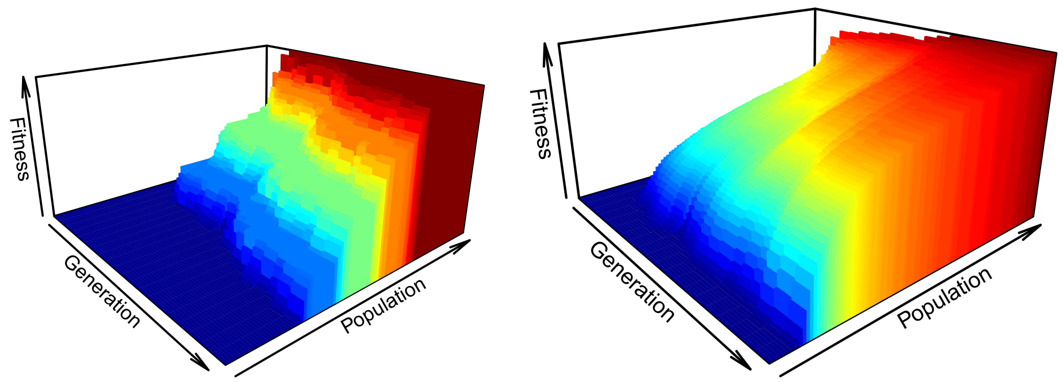
The fittest agents from each run were evaluated in the Robustness Test:

$$g(G, n) = \frac{\sum_{i=1}^{10^4} f(E_n^i, G)}{10^4} \tag{4.1}$$

where g represents the fitness function of the robustness test, taking a genotype (G) to evaluate and river size (n). f is the fitness function of the task, which takes RC world (E) with appropriate river size (n). 10^4 is for the static RC world configurations.

Comparing figure 4.5 and 4.7, it demonstrates that training performance does not translate to general-purpose performance. RC worlds with river width 0 and 1 achieved a consistent completion (93-94%) with the fitness modification, with and without delta. Whereas, the original fitness function's performance saw a larger deviation and spread across a lower completion range (80-87%). Then, RC worlds with river width 2 see a drop in performance on all approaches, and a subsequent drop at river width 3. Table 4.3.1 displays the p values from a two sample t-test between delta and non delta approaches. When using the fitness modification, delta inputs provide a significantly higher completion percentage during the more complicated RC worlds (river width 2 & 3). All approaches with the fitness modification saw a statistically significant advantage, compared to their counterparts without.

Evolutionary runs were extended to higher generations to assess if fitness had plateaued in general-purpose performance. This is due to the fact that in figure 4.5



(a) Original RC Fitness Function *(b) Fitness Modification*

Figure 4.6: **Representation of HyperNEAT's fitness score in a population without (a) and with (b) the fitness modification.**. The fitness modification being the mean of 10 RC world runs. Simulated over 200 generations, a maximum fitness of 4 and population size of 1000.

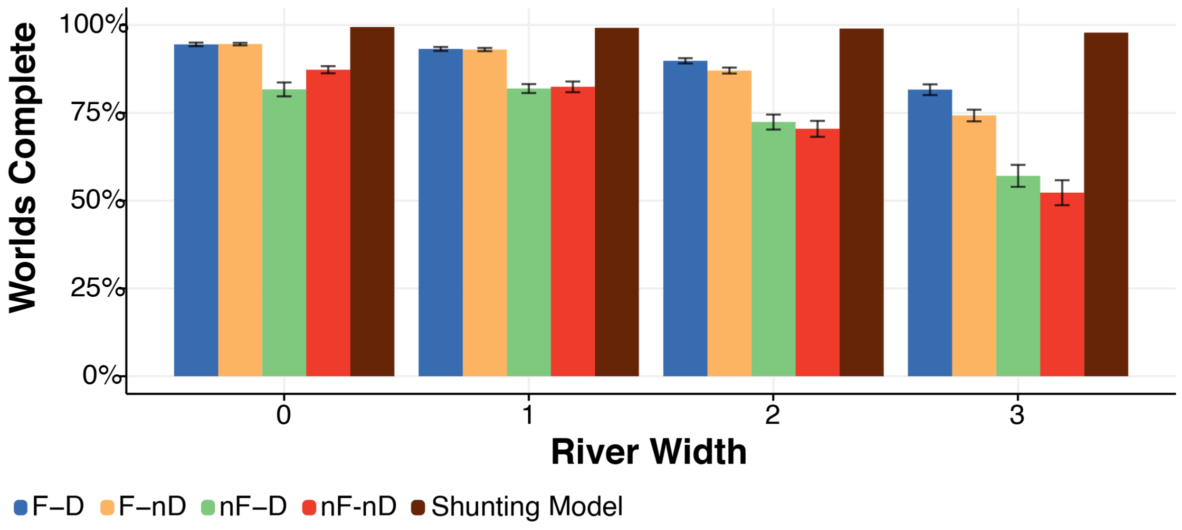


Figure 4.7: **Completion rates of Robustness Test, using each HyperNEAT approach.** Individuals simulated are the fittest at generation 200 from all runs, twenty-five evolutionary runs, per approach, per river width. Error bars represent standard deviations.

we see fitness maintaining a high level of consistency in training to achieve fitness 4, however figure 4.7 shows this does not translate to general performance. By extending the runs, gains may be achieved in general performance that is not visible in training

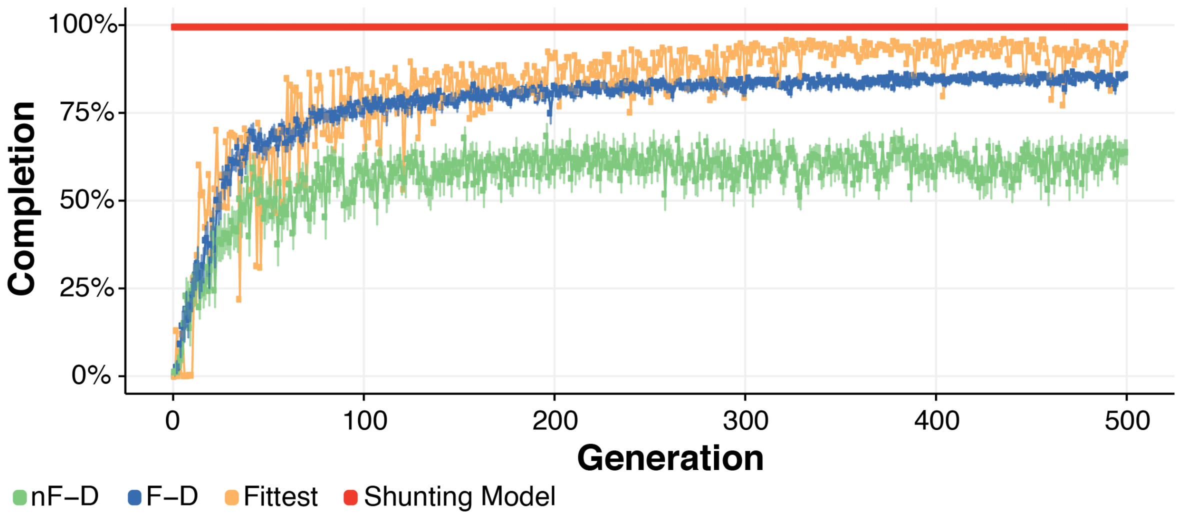


Figure 4.8: **Completion rates of Robustness Test at river width 3 using HyperNEAT with and without the fitness modification.** Twenty-five evolutionary runs, per approach, for 500 generations. Error bars represent standard deviations.

Two Sample T-Test (p values)

	River 0	River 1	River 2	River 3
		F-D		
F-*n*D	0.8902	0.809	**0.01796**	**0.002184**
		***n*F-D**		
***n*F-*n*D**	0.01862	0.806	0.5363	0.3156

Table 4.4: ***p* values from two sample t-tests between CPPN delta and non-delta approaches in RC Robustness Test.** The original fitness function and fitness modification are used. Individuals simulated are the fittest at generations 200.

performance. Figure 4.8 displays the mean completion in the Robustness Test at river width 3 with an extension to 500 generations. Performance for each fitness function appears to stay relatively consistent with results achieved at the original generation 200. However, the fittest individual achieved its highest performing completion at generation 356 and after 300+ generations the performance appears less volatile than in the preceding generations. The original fitness function appears noisy in general-purpose performance. In contrast, the fitness modification can produce greater consistency at a higher completion rate and in turn produced the fittest individual.

Individuals with high completion percentages in the Robustness Test produced similar functioning activity landscapes. However, none resembled the SM. Figure 4.9 shows an example of activity landscapes produced by an individual with a 95.9% completion on the Robustness Test at river width 3. Agents which require stones are directed south of the river by the activity landscape. Once a stone is carried, agents are directed to the furthest north position with a peak at the resource's location. This forces the agent north until interaction with the river, at which point the stone is placed upon the river. Agents will then again return south to acquire stones, and continue until a bridge is complete.

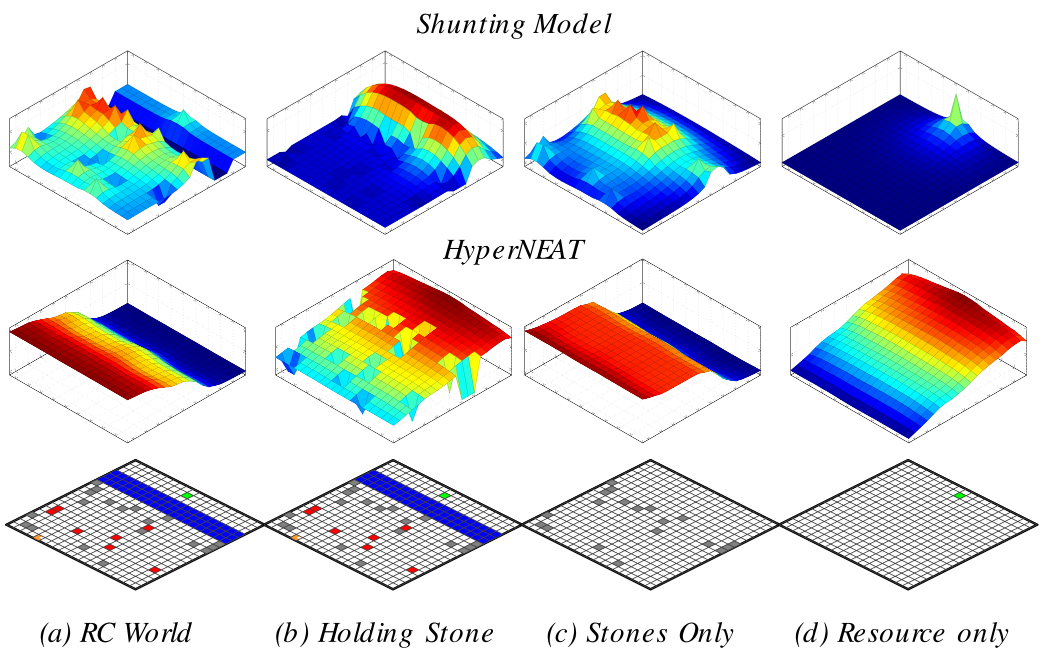


Figure 4.9: **RC worlds (below) with their corresponding activity landscapes (above).** Each activity landscape is produced by the fittest individual in the Robustness Test with the Shunting Model and HyperNEAT.

4.3.2 Discussion

No HyperNEAT implementation could produce the same quality of deliberate, robust motion planning when compared to the SM at any river size. Solutions at river width 0 and 1 produce relatively consistent results, but a performance degradation occurs at river width 2 and there is a further drop at river width 3. Beyond river width 1 the task requires new behaviours: such as; the deliberate movement from river to stone after a stone placement and constructing a connecting bridge. Each increment in river width correlates to the need for greater sophistication in both these behaviours. Therefore, the drop in performance suggests HyperNEAT solutions lack these deliberate planning skills. However, practical HyperNEAT implementations are possible. Individuals with the fitness modification and delta inputs were found achieving 90%+ RC world completions on the most difficult tasks.

Figures 4.7 and 4.8 shows that the fitness modification provides superior general-purpose performance. The modification allows the obvious benefit of multi-performance feedback. Figure 4.6 also shows the benefit during training. In comparison, the original RC fitness function only provides five fitness states for diversity to occupy. A relatively large amount of the population then remains at fitness 0 throughout. The fitness modifier provides a larger impact overall when compared to the delta counterpart. However, when used in combination there is a statistical advantage in the more difficult tasks.

Delta inputs and the fitness modification show an advantage at river width 2 and 3, as seen in table 4.3.1. At these river widths, agents require greater deliberative behaviour. The benefits of deltas may be related to creating relational patterns around activated inputs, therefore providing contextual awareness of nearby objects such as stones. Figure 4.10 shows how a highly robust CPPN utilised the delta inputs. Absolute inputs could theoretically also achieve these patterns, albeit at greater difficulty and evolutionary cost. However, just how deltas provide an advantage is not clear from observing successful activity landscapes, as solutions appear to rely on linear patterns.

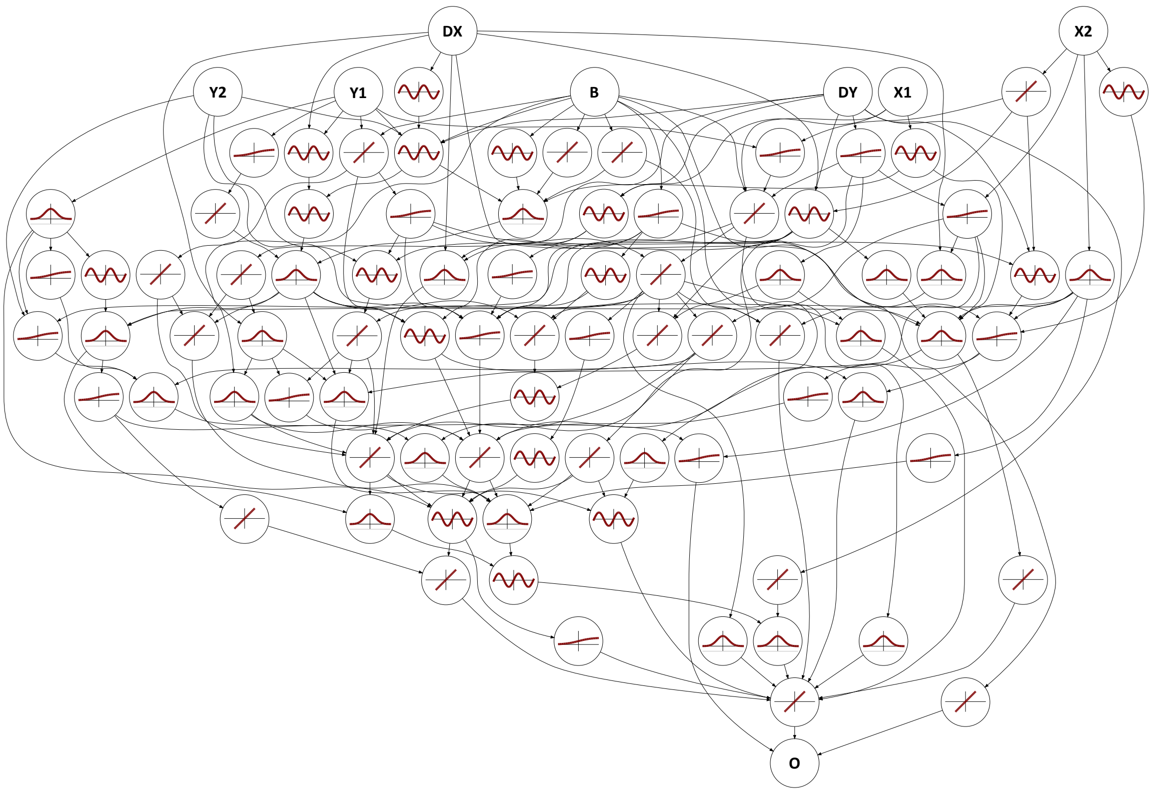


Figure 4.10: **Illustration of a highly robust CPPN for the RC Task.** Where $X1$,$X2$,$Y1$ and $Y2$ represent their appropriate inputs; DX and DY represent deltas for x and y, respectively; B represents the bias and O is the output. Activation functions include sigmoid, sinusoid, gaussian, and identity.

Successful HyperNEAT solutions exhibit a 'funnelling' type behaviour that was proven reliable for general-purpose performance on the Robustness Test, as shown in figure 4.9. At the simplest interpretation, an agent will move north or south depending on whether a stone is being held. Figure 4.9 (a and c) shows that when stones are desirable evolution has discovered being south of the river is beneficial for locating stones. The highest activation points are those furthest south. Once a stone is placed on the river and another has to be collected, the further south an agent moves the greater its likelihood of interacting with another stone. Figures 4.9 (b and d) show how an agent will traverse to the resource once a stone is obtained. Despite positive general-purpose results on river widths 0 and 1, there is a problem in practice, as agents' movement lack the compelling behaviours of believable, deliberate motion planning required for robust performance on wider rivers.

Agents take complex, longer paths to stones in the RC world while ignoring those in their immediate neighbourhood. This can lower agents' success due to their 100 steps limit, but more importantly it appears to show a lack of intelligent or deliberate decision-making. This is most common after the completion of a bridge, since agents will proceed back to a stone before obtaining the resource. In comparison, the SM produces a local activity field around the resource that allows agents to proceed toward it once the bridge is complete. Agents from the RC 3D simulation in Stanton and Channon are able to exhibit life-like responses, due to their reaction to the activity space [222], which observers have described as resembling surprise, confusion, and even happiness. HyperNEAT's fittest model may lose these subtle behaviours if it were to replace the SM in the RC 3D system.

4.4 Conclusion

This chapter demonstrates HyperNEAT's capability to produce feasible deliberate and robust motion planning, but, for this task, not to the quality of a pre-designed solution. The chapter's preliminary experiment was designed to examine the performance of both direct and indirect encoding on producing scalable motion planning using a form of artificial potential fields. NEAT was shown that in any form, it would not be capable of producing the motion planning this task requires. This is apparent by NEAT losing statistical significance to a random matrix of values as the world size increased. Also, NEAT was achieving inconsistent results at very small environment sizes (4x4). So, NEAT will not be utilised further and did not make it to experiments on the original RC world size (20x20). However, HyperNEAT showed promise by achieving the highest possible fitness in all RC world sizes consistently; including the original RC world size.

A new experiment setup was established to further analyse HyperNEAT's performance in producing motion planning. These further results were achieved using a general-purpose HyperNEAT configuration with no problem-specific aspects of the network design. The performance maintained a great level of consistency (90%+) in locating the resource and simple bridge-building with a multi-evaluation fitness function, shown in figure 4.7. Mean performance drops to a respectable level (80%) at the most difficult deliberate task, but individuals can still be found 3-4% below the performance of the SM, as shown in figure 4.8. This chapter also demonstrates the importance of relative distance information in producing greater general-purpose solutions at more difficult deliberate tasks.

However, HyperNEAT contains caveats which means that it will no longer be pursued for the RC task. Firstly, HyperNEAT is dropping a noticeable percentage in robustness before the introduction of more significant challenges such as; the avoidance of traps and training the DN simultaneously. Next, due to the way the genome creates weights for the substrate, a second evolutionary process would be needed to simply evolve the DN. Although it would be ideal to solve the task with a single network, at this stage, we do not see an obvious solution. Therefore, we shall incrementally move towards the goal. Incremental changes require the use of the DN, as a replacement SM only operates with the pre-processing stage of the DN. So instead, a solution which utilises a single genome which controls all networks will be advantageous at this stage. The next chapter will introduce a novel use of recurrency in a convolution network in an attempt to overcome these hurdles.

5| Evaluating Motion Planning with a Recurrent Convolutional Network

5.1 Introduction

In this chapter, a new solution is conceived to overcome the issues experienced in motion planning with HyperNEAT. Although HyperNEAT was able to produce a motion planning system that could complete the RC task at its most difficult challenge, it was not as robust as the original static solution (the SM); robustness being how well the solution adapts to general environments. This is before the more difficult task of including the DN into the evolutionary process. As a result, HyperNEAT does not seem suitable as a replacement for the SM. Instead, this chapter will introduce a novel system using ConvNet; taking inspiration from deep ConvNets, which in recent years have demonstrated that advanced behaviours are obtainable without careful engineering and considerable domain expertise.

ConvNet's kernel focuses on local areas of an input feature and mimics the SM's reliance on local connections. The ConvNet architecture presented here is unorthodox as instead of multiple layers of convolution, this architecture aims to achieve this in one layer with the use of recurrent links. Convolution applies as in any other standard ConvNet, but the original feature will continually be incorporated back into the result with the use of the recurrent links. So, the only evolvable aspect of the model is the kernel weights. The kernel iteratively scans across the input multiple times and the output represents an activity landscape for the RC Task. A reason that kernels are a beneficial aspect of ConvNets is that it condenses the genotype to a small static value which would not increase with the input layer size; which is an issue with a standard ANN (section 4.2.2). Further, the use of a single kernel shares a single rule on how values in the input disperse from local areas which is similar to the SM with its localised static connections.

Finally, the output of the ConvNet is required to keep the same spatial size as the input, by using a single layer with recurrent connections it is not necessary to create multiple layers of same spatially sized convolutional layers. This would cause a greater number of kernels, which in turn causes a larger genotype and potentially conflicting rules in the way kernels create pathways in the activity landscape. In this chapter, reactive behaviour is incorporated into motion planning by allowing obstacle avoidance; this was averted in previous models due to the complexity. Obstacle avoidance is an agent's ability to avoid harmful objects in the environment while traversing to positive ones. This will be evaluated with a simple GA, not utilising the more complex and beneficial diversity mechanisms that was discussed in chapter 2.

5.2 Experimental Setup

In these experiments, the functionality of the traditional shunting equation was replaced with a shallow ConvNet architecture; in which a single layer and single kernel are used. As in the last chapter, this network will replace the SM to produce an output that forms an activity landscape that an agent can traverse to achieve long term deliberate motion planning. Two sets of experiments were carried out for 100 runs. The first evolved the DN while the SM remains static. Next, the ConvNet-SM was evolved with a static DN; the static DN was pre-evolved to achieve the highest fitness on the RC Task with the original SM [192].

5.2.1 RC Task

The RC Task contains the same alterations in previous chapters [107]. So, fitness aggregates over 10 RC Task attempts and successful agents are subject to the Robustness Test. This simulates agents through 10^4 static RC world configurations with a river width of 3, the most difficult type of world this task offers.

5.2.2 Network Architecture

The architecture introduced in this chapter replaces the SM from Robinson et al. [192] with a single layer ConvNet with recurrent connections, see figure 5.1. The ConvNet uses a 3x3 kernel with a stride of 1 and padding of 1, to synchronously update activities in the same layer. After each convolution iteration ReLU applies across the layer. ReLU

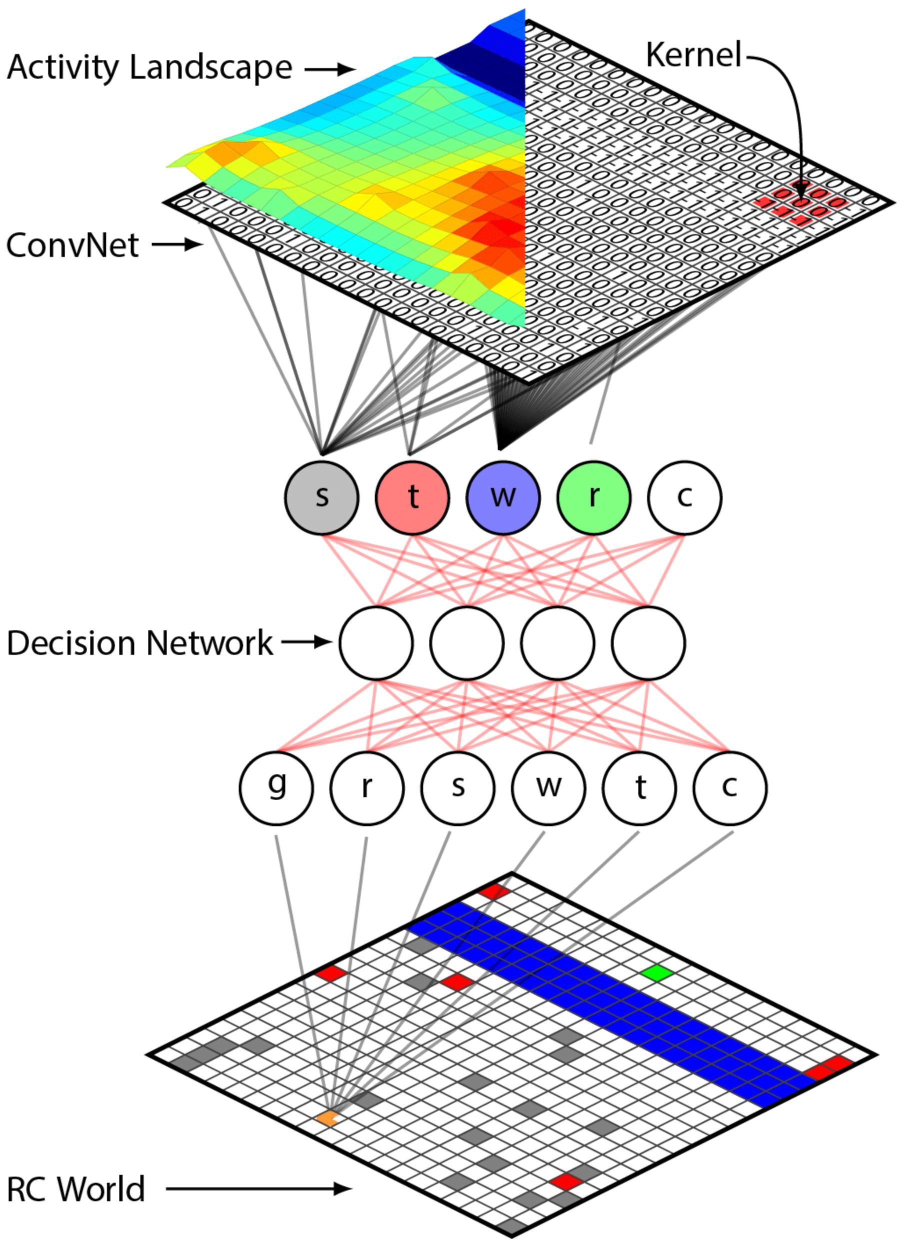


Figure 5.1: **Neural architecture for the RC Task with recurrent ConvNet.** Attributes at the agent's position (g=grass, r=resource, s=stone, w=water, t=trap, c=carrying) determine inputs to the Decision Network. Attributes of the RC World are converted to iota values via the Decision Network outputs and mapped to the corresponding position of the attributes. This is a static pre-processing stage to prepare the inputs for the ConvNet. A 20x20 array of iotas are passed to the ConvNet layer. A 3x3 kernel is passed across the layer to perform recurrent convolution, synchronously updating activities in the same layer. The network is activated a fixed number of times. At each activation the kernel appends the previous output from the kernel to the current DN output. The activity landscape is a visual representation of the output after completion. Links in red represent evolvable weights.

typically learns much faster in networks with many layers, shown in [68], which is not relevant due to our networks size. However for this experiment, the ReLU has the added benefit of replicating the shunting equation's ability to propagate only positive values. Each update, the original iota values at iteration 0 are incorporated back into the 20x20 output of the ConvNet via an identity kernel passthrough. This retains the positions of the original iota values, while propagating activity values across the layer via multiple activations. The input values are referred to as iotas, but after activation, the output values are referred to as activity values; to represent the activity landscape the values represent.

The activity landscape can only function if the ConvNet's output resembles the same dimensions as the input. Thus, the lack of pooling and use of padding is necessary in order for the output to remain the same spatial size. At each time-step the network incorporates the current agent cell to inform the desirable and undesirable objects in the landscape. The network initialises each world step and then activates multiple times via a fixed iteration number. The RC world size instructs the iteration number value; for this work the value is 50. The ConvNet output represents the activity landscape. In the landscape, an agent follows the activation of its highest surrounding neighbour (gradient ascent). The genotype sizes for the DN and ConvNet-SM configurations are 44 and 9 respectively. The DN's topology is described in section 3.2.1, and the ConvNet-SM evolvable weights are each position in the 3x3 kernel. All weights are bound between -1 to 1. This process can be described in the following equations:

In terms of activities:

$$G^{new}[m,n] = \sum_j \sum_k (g[j,k] \underbrace{e(I[m+j,n+k])}_{\substack{\text{regular} \\ \text{convolution}}} + h[j,k] \underbrace{f(G[m+j,n+k))}_{\substack{\text{recurrent} \\ \text{convolution}}} \tag{5.1}$$

In terms of output o and p:

$$o[x,y] = e(I[x,y]) \qquad p^{new}[x,y] = f(G[x,y])$$

$$p^{new}[x,y] = f(\sum_j \sum_k (g[j,k]o(m+j,n+k) + h[j,k]p[m+j,n+k])) \tag{5.2}$$

where I and G are activity values, I from input and G being the recurrent layer; e and f activation functions, Identity and ReLU respectively; $e(I)$ and $f(G)$ are output values (i.e the activity values after activation functions are applied), and stored in o and p; g and h are kernels, where h is the kernel with evolvable weights and g is the identity kernel (i.e. all values 0, except the centre value of 1); j and k being the row and column of the filter size; m, n, x and y are row and column values. A visualisation of this process can be seen in Figure 5.2. The inputs for the kernel are unbounded (i.e $(-\infty, \infty)$).

5.2.3 Genetic Algorithm (GA)

This model uses an extremely simple GA with a population of 100 individuals, representing neural network weights. In each generation, every individual's performance on the RC Task is assigned a fitness value. Elitism is applied, storing 10% of the fittest individuals for the next generation. The remaining population is generated via single-point crossover and mutation, with parents selected at random from the previous generation. Each off springs genome has a 25% chance of mutating a single parameter with an additive Gaussian noise value. Once the highest fitness has been achieved evolution is stopped and the agent is evaluated on the Robustness Test. If an agent does not achieve the highest fitness it is considered a failed attempt.

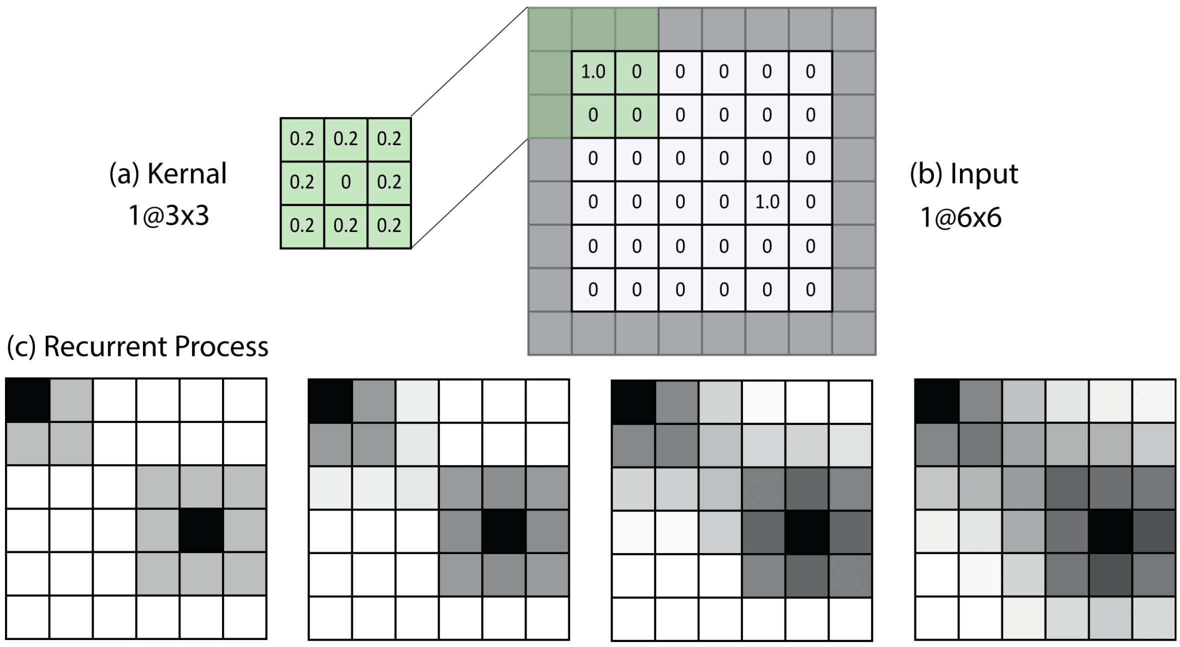


Figure 5.2: **A visual example of the recurrent convolution process.** The input to the network consists of an 1 x 6 x 6 image followed by a recurrent convolutional layer containing a 3 x 3 kernel. At each iteration the kernel is applied to each cell in the input (a stride of 1 and padding are used to match the input to output). Each iteration, the original iota values at iteration 0 are incorporated back into the 20x20 output of the ConvNet via an identity kernel passthrough.

5.3 Results

100 runs for 10^4 generations were carried out with each strategy on the RC Task. During training, if agents were unable to complete all 10 RC Tasks to its most difficult behaviour they failed the run.

- **Static-SM** - where the DN evolves and the SM follows the shunting equation.
- **ConvNet-SM** - where the DN is pre-evolved and the SM evolves via the kernel of the ConvNet. The static DN was pre-evolved to achieve the highest fitness on the RC Task with the original SM [192].

Set	Best	Worst	Mean	Stdev	Success
Static-SM	1	138	43.58	30.73	100%
ConvNet-SM	43	4146	845.59	845.11	100%

Table 5.1: **Best, worst, and mean number of generations required to complete the RC Task.** Completion is defined by any agent's ability to complete the RC Task at the hardest difficulty 10 times.

Table 5.1 demonstrates that every strategy was able to complete the RC Task to the highest level of difficulty. As previously established, Static-SM achieved the highest standard of reactive and deliberative behaviours in all runs, this is also true for ConvNet-SM; both achieve 100% success in every run. However, all evolvable motion planning strategies on average took more generations to find a successful solution.

Agents that achieved completion during each run were evaluated in the Robustness Test with river width 3. This simulates agents through 10^4 static RC world configurations. Agents performance on this test represents their ability to adapt to general-purpose environments. Figure 5.3 presents these results as well as those from the HyperNEAT architecture used in the previous chapter [107]. HyperNEAT and ConvNet-SM both utilise the same pre-evolved DN for direct comparison.

All strategies are statistically distinguishable from each other in their general-purpose completion ratings, established via a two-sample t-test with p values less than 0.05. Static-SM still provides the most consistent results. ConvNet-SM provides comparable results with only a slight increase in mid-spread and a slightly lower average of 98.97% vs the 99.96% of Static-SM. In the same task ConvNet-SM exhibits greater performance compared to HyperNEAT.

Another appealing aspect of the shunting equation is the efficiency of agents' move-

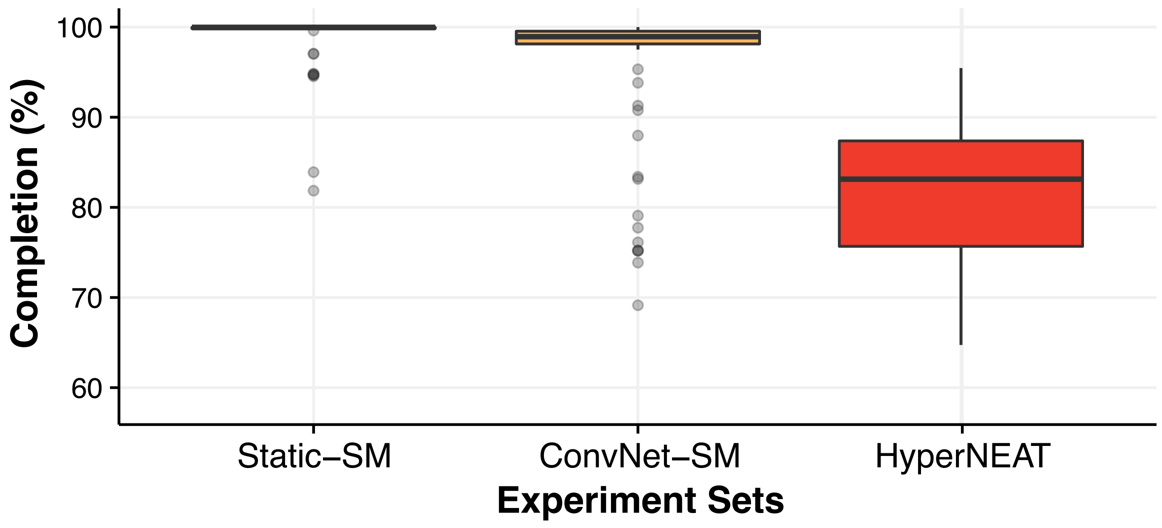


Figure 5.3: **Completion percentage of Robustness Test across all strategies aggregated over 100 runs each.**
(Static-SM = static shunting model. ConvNet-SM = evolvable shunting model.)

ment, wherein agents move to the closest desirable object. As established previously, without efficient movement the actions of agents can seem undeliberate and unintelligent. So, the new architecture has to be evaluated in this area. The age of an agent after completion of an RC world is a good metric for judging how efficient an agent traverses the environment. Figure 5.4 aggregates the average age of agents after the completion of an RC world on the Robustness Test. Results are limited to those that achieved 98% or above on the RC Robustness Test. Restricting to high completions provides an accurate comparison between results. HyperNEAT was not included as no runs achieved a completion percentage above 98%. As seen in figure 5.4, the Static-SM stays consistent with an average age of 45.2. Yet, the ConvNet-SM is able to produce a lower average of 43.43. Via a two-sample t-test, results show a statistical significance of ConvNetSM over the Static-SM ($p < 0.05$) in their ages on the Robustness test.

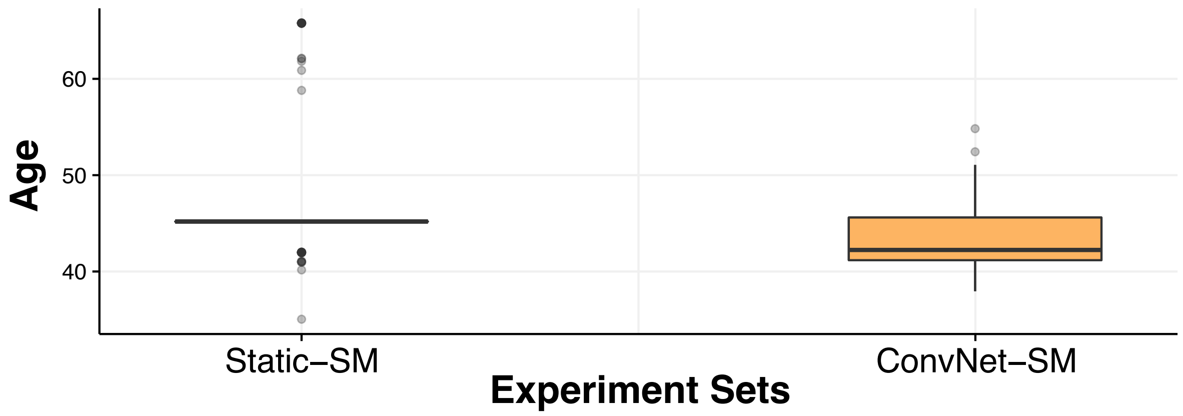


Figure 5.4: **Mean age of agents after a successful RC world completion in the Robustness Test.** All strategies are showing results from agents with 98% completion or higher in the Robustness Test.
(Static-SM = static shunting model. ConvNet-SM = evolvable shunting model.)

ConvNet-SM also has a lower spread than the Static-SM but only the Static-SM was able to achieve the lowest average age of 33.05.

Figure 5.5 provides a representation of the activity landscapes of Static-SM and ConvNet-SM on the same RC world. An example was purposely chosen in which the ConvNet-SM has an age advantage over the Static-SM; this was done to provide context to the improved average age. Viewing the RC world after completion of the Static-SM shows two attempts at a connecting bridge. In motion, agents using Static-SM locate stones close to the river and prematurely place the stone while traversing forward. This is due to the shunting equation forcing agents forward without enough room to return to the centre of the river. If stones are further from the river agents funnel to the centre, as can be seen in the activity landscape when an agent is holding the stone. This is evident again when comparing stone locations from Static-SM to ConvNet-SM; those closest to the river in Static-SM have been used to attempt an unsuccessful bridge.

5.4 Discussion

Compared to static-SM, ConvNet-SM was able to produce equally deliberate and robust motion planning in even the most difficult RC worlds, while on average achieving more efficient planning. The static-SM still provides superior reliability in completion. Although, ConvNet-SM is an inherently more difficult task and degradation to completion average is minimal. In comparison to HyperNEAT, the average ConvNet-SM completion was higher than HyperNEAT's fittest agent. Further, ConvNet-SM in this experiment did not dismiss negative iota values like in the HyperNEAT architecture. ConvNet-SM benefits from using a ReLU to avoid the propagation of negative values. Thus, ConvNet-SM is capable of replicating the shunting equation's functionality.

The evolvable SM demonstrated an unforeseen, novel advantage over a purpose-built system. When compared to the shunting equation, agents trained with ConvNet-SM achieved instances of the highest completion rate as well as a superior average age on the Robustness Test. Analysis of the static-SM shows RC worlds with stones close to the river forced agents to create bridges in an unstructured manner. By contrast, ConvNet-SM created deliberate bridge designs despite the location of the stones, see figure 5.5. Both evolvable strategies have activity landscapes which are difficult to interpret, thus the kernel weights were examined. ConvNet-SM creates pathways to positive values via diagonal paths, whereby the corners are the most dominant of the 3x3 kernel. In motion, agents do not suffer from the long and questionable choices in the movement which appears with HyperNEAT. This is quantified when comparing ages in the robustness test from the fittest ConvNet-SM (43.74885) and fittest HyperNEAT approach (68.7384) being significantly different ($p <$ 2.2e-16); while ConvNet-SM achieves greater competition performance (99.98% vs 95.9%). A highly pre-evolved ConvNet-SM could replace the SM in models that use it. This would allow superior motion planning without changing the network architecture or training.

For future work, there are established working examples of greater general-purpose architectures this work can adapt. Currently, agents follow the highest surrounding iota in the activity landscape. Work in Stanton and Channon [222] used a fully connected feed forward neural network to allow evolution to discover the relationships between iota values and movement; then, the outputs provide direct control over the agent's movement. Work in Borg and Channon [21] generalised the DN to use RGB values as inputs, allowing greater abstraction from task-specific interactions. This also allows a fixed DN despite a varying number of object types in the world. As these considerations are addressed, the network may adapt to more complex scenarios and environments.

5.5 Conclusions

This chapter has shown that a shallow ConvNet, with a novel recurrent process, is capable of producing deliberate and robust motion planning to the quality of a pre-designed solution, with greater efficiency; these results were achieved with an extremely simple GA. Individuals were exposed to a task of scaling complexity and were required to complete the task at its most difficult behaviour multiple times. The use of a multi-evaluation fitness function in training, as a stopping criteria, encourages the evolution of motion planning that is adaptable to a variety of different world combinations.

Compared to a static motion planning variation (the SM), the single layer ConvNet can achieve an average general completion that is within 3% on a difficult deliberate task. This architecture could seemingly adapt to other tasks designed in the same manner; or a replacement for a motion planning system that uses an artificial potential field. The main benefit of this approach was the superior deliberate bridge building achieved in the RC Task. This results shows a potential for further unforeseen, novel

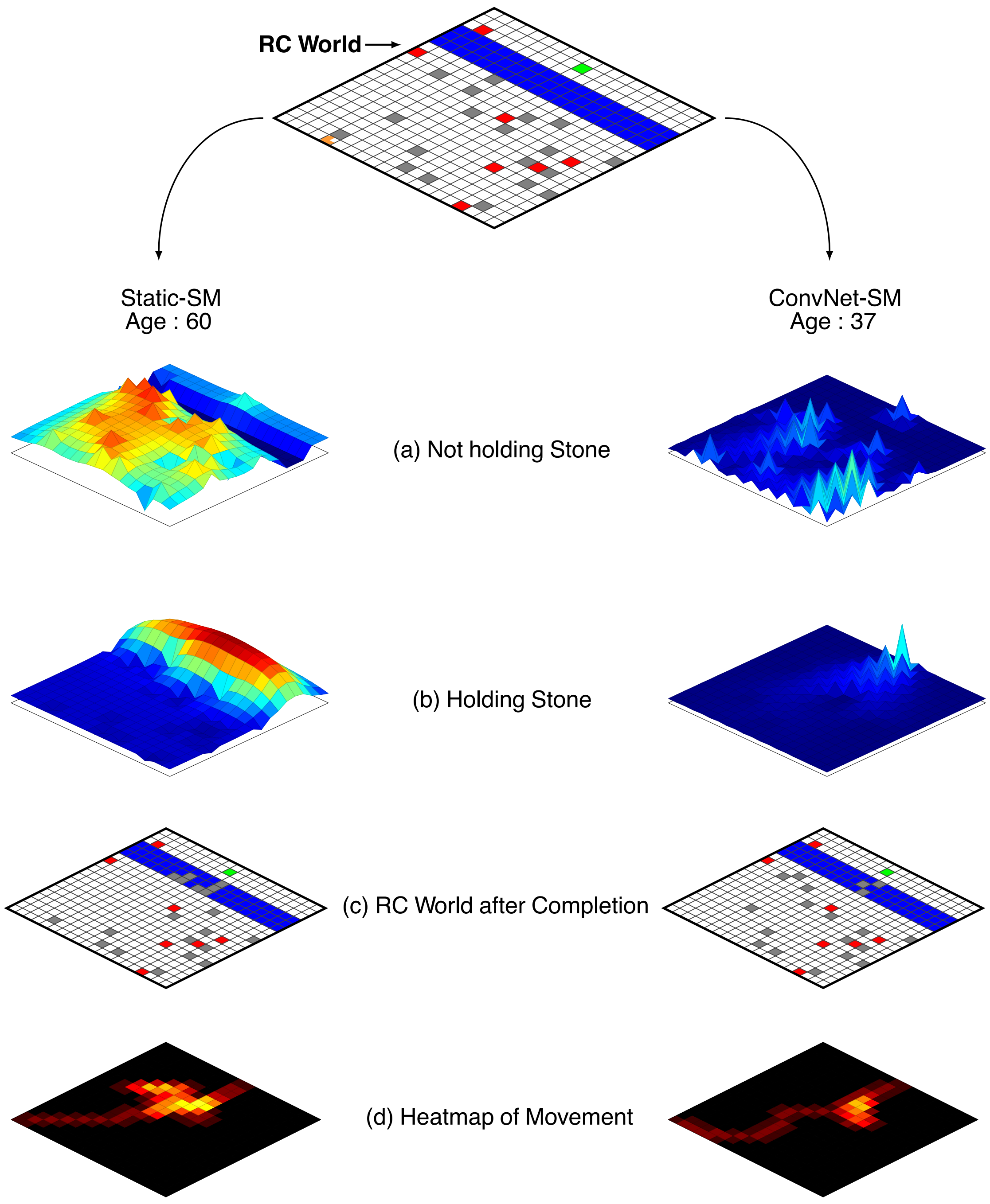


Figure 5.5: **RC world with the corresponding activity landscapes for each strategy.** Each strategy is labelled and given the age of the agent after completion. Each strategy has a completion of 98+% in the Robustness Test.

benefits when removing model restrictions.

This thesis can now continue the theme of removing task-specific aspects in the RC Task, in favour of greater control from the network. Due to the positive increase in motion planning efficiency, the removal of more task-specific aspects may result in additional benefits, as well as an architecture that can adapt to greater tasks. The next chapter, will use the recurrent ConvNet in combination with the DN, as well as new networks which aim to further remove domain specific information.

6 | Generalised Neural Network Architecture for Long-Term Planning with Sparse Rewards

6.1 Introduction

An aim of the work in this thesis has been to retain the long term deliberate planning in the RC task's architecture, while continually generalising it. Chapter 5 has now established that a recurrent ConvNet can produce effective evolvable motion planning in the RC Task with greater efficiency than a static implementation. Evolvable motion planning removed a static behaviour constraint, but now, it has to be validated whether it can integrate into models of greater evolutionary complexity; so, now greater experimentation can take place. Two of the main constraints on the previous model have been the DN as it was (1) pre-evolved and (2) domain-specific.

The pre-evolved DN was evolved with the static SM until the best behaviour for the RC task was found. This allowed the previous experiments to purely assess motion planning capabilities, but now the DN has to be incorporated back into the evolutionary process to examine whether both networks can evolve simultaneously. However, the DN's topology has to be changed to remove the domain-specific elements; this is referring to the inputs for the DN (section 3.2.1) are predefined based on the objects in the RC Task world. As a result, the network architecture is constrained to this specific task and if it were to change in a simple way, such as an additional object, the architecture would also have to change (i.e. an additional input and output). This makes the current DN unrealistic to deploy in general two-dimensional environments as drastic topology changes will also require changes in the mutation rates. This would cause a trial-and-error process and each environment will require its own considerations. So, in this chapter a DN is proposed which uses pixel data for inputs, so the topology no longer changes when the environment does.

Another aspect that has remained static in the RC Task is the gradient ascent rule, in which agents follow the steepest gradient in their Moore's neighbourhood. Although there is no inherent benefit to replacing this behaviour in this current task, the three-dimensional implementation of the RC task has found success in not dictating where the animat should move but instead passing activity values via sensory information. In a three-dimensional environment with continuous space, as opposed to discrete positions, a rule like gradient ascent can not easily be applied. But, removing the rule at this stage, in favour of a traditional ANN, greater evolutionary complexity is put on the model to find the appropriate long term deliberate planning behaviours; and, as a result, the network architecture becomes more generalised.

The experiment setup remains the same as detailed in 5.2, with only the neural network architecture being modified. Incremental steps will be made towards the fully generalised architecture, as a result, the new and old network topologies will be paired with each other in a modular setup to allow the strengths and weaknesses of each approach to be evaluated. The proceeding sections will detail these changes.

6.2 Generalised Neural Network Architecture

This experiment removes domain-specific designs (features) from each sub-network in the modular architecture. For clarity, domain-specific designs are those which tailor the network to the RC Task. When removed, the revised architecture should be suited to general-purpose 2D environments. The two sub-networks are the Decision Network and the Activity Network. In addition, we introduce a new neural network, called the Movement Network whose role is to decide how the agent will move based on the normalised activities of the cells in its neighbourhood (Figure 6.2.3). Modified designs are proposed for each while explaining the previous limitations and how the modifications address them.

6.2.1 Generalised Decision Network

Each object in the environment has a discrete representation in the Decision Network for both input and output. This is advantageous as it allows a direct relationship to the significance of an object. However, for a general-purpose network this would not be possible as (1) objects in the environment may be unknown at run-time (i.e. emulators) and (2) having very many objects would result in an overly large network unsuitable for direct evolution. Other techniques could be utilised here such as automatic object detection but the approach in this chapter is to use pixel data for inputs. The environment has been modified to allow cells to have colour and the agent's decision network now can read the red, green, and blue values from the cells. The network topology uses 4 inputs, a first hidden layer of size 10, a second hidden layer of size 8 and 2 outputs. 3 inputs are from the pixel colour channels. The final input is for additional environmental information. For example, in this chapter, it is whether the agent is holding an object. One output is the iota value. This is no longer limited to 3 discrete values but is instead continuous from -1 to 1. The other output is an action output; in this chapter, whether an agent should pick up or put down an object. The topology of the DN was chosen by trying out various topologies and seeing which topology gave performance matching the pre-evolved decision network discussed in earlier chapters. This Pixel Decision Network operates differently to the previous implementation. Objects in the environment now hold a colour value. The colour of a cell is determined by the colour of the object residing in the cell. Each cell is visited and its colour value is fed through the network. An iota value is generated and is stored in a separate 20x20 matrix of iota values. The x and y position of the iota replicates the object's x and y position in the RC World. The grid of iota values are passed to the Activity Network.

6.2.2 Generalised Activity Network

The Shunting Model uses a static equation to disperse activity values across a landscape, which creates predictable and consistent motion planning no matter the environment. However, this behaviour may not be suited to all tasks. Instead, evolution should dictate the appropriate motion planning behaviour. To make this possible, a solution is needed that can adapt to variable and relatively large input sizes. In the RC Task, for example, a fully-connected 20x20 network could not be feasibly evolved due to the number of connections. Instead, a ConvNet with recurrent connections is used. ConvNets have been proven useful on a wide array of computer vision tasks and can be encoded on small genomes, as only kernels need to be represented. In this chapter, recurrency refers to kernels taking inputs from the same layer that they output to. Each iteration, the original iota values at iteration 0 are incorporated back into the 20x20 output of the ConvNet via an identity kernel passthrough. The idea behind this is to retain the features of the original input and focus on the propagation of values, rather than on creating new features. After many iterations, values propagate across the landscape from the original iota positions.

The ConvNet uses a 3x3 kernel with a stride of 1 and padding of 1. Updates apply synchronously in the same layer. After each iteration, ReLU applies to outputs and therefore only propagates positive values. The activity landscape has the same dimensions as the RC world. So, pooling is removed and padding is necessary to retain spatial size. In this chapter, 50 iterations of recurrent convolution are carried out for each update of the Activity Network. Previously, an agent would move to the neighbouring cell with the highest activation. Now, the activity values of the neighbouring cells (those surrounding the animat) are given to the Movement Network.

6.2.3 Generalised Movement Network

Previously, an agent moved to the cell with the highest activity value in their neighbourhood. Intrinsically, this behaviour requires the Activity Network to create paths to desirable objects. But, with the evolvable Activity Network, the function should be malleable to allow new behaviours. Thus, an evolvable Movement Network is needed to remove preconceptions of the Activity Network's role. The topology has 8 inputs and 8 outputs. Inputs are the normalised activities of neighbouring cells from the Activity Network. The agent's occupying cell is not included as there is no benefit to remaining and a cell. Further, as the DN inputs will remain the same, the same behaviour will continue each world step, so a loop occurs in which the agent will not move until death from age occurs. A MAX operation is applied to the outputs; this leaves a single active output cell. Outputs correspond to movement directions; the agent moves in the direction of the highest-activity output.

6.2.4 Genetic Algorithm

This experiment uses an extremely simple GA. An individual's genome consists of neural network weights. The population size is 1000. In each generation, each individual is evaluated on the RC Task 10 times and assigned the mean performance over these runs as its fitness score. Elitism applies to the population, retaining the top 10% fittest individuals. Random individuals are then taken from the full population range to generate offspring. Offspring are produced via single-point crossover and mutation.

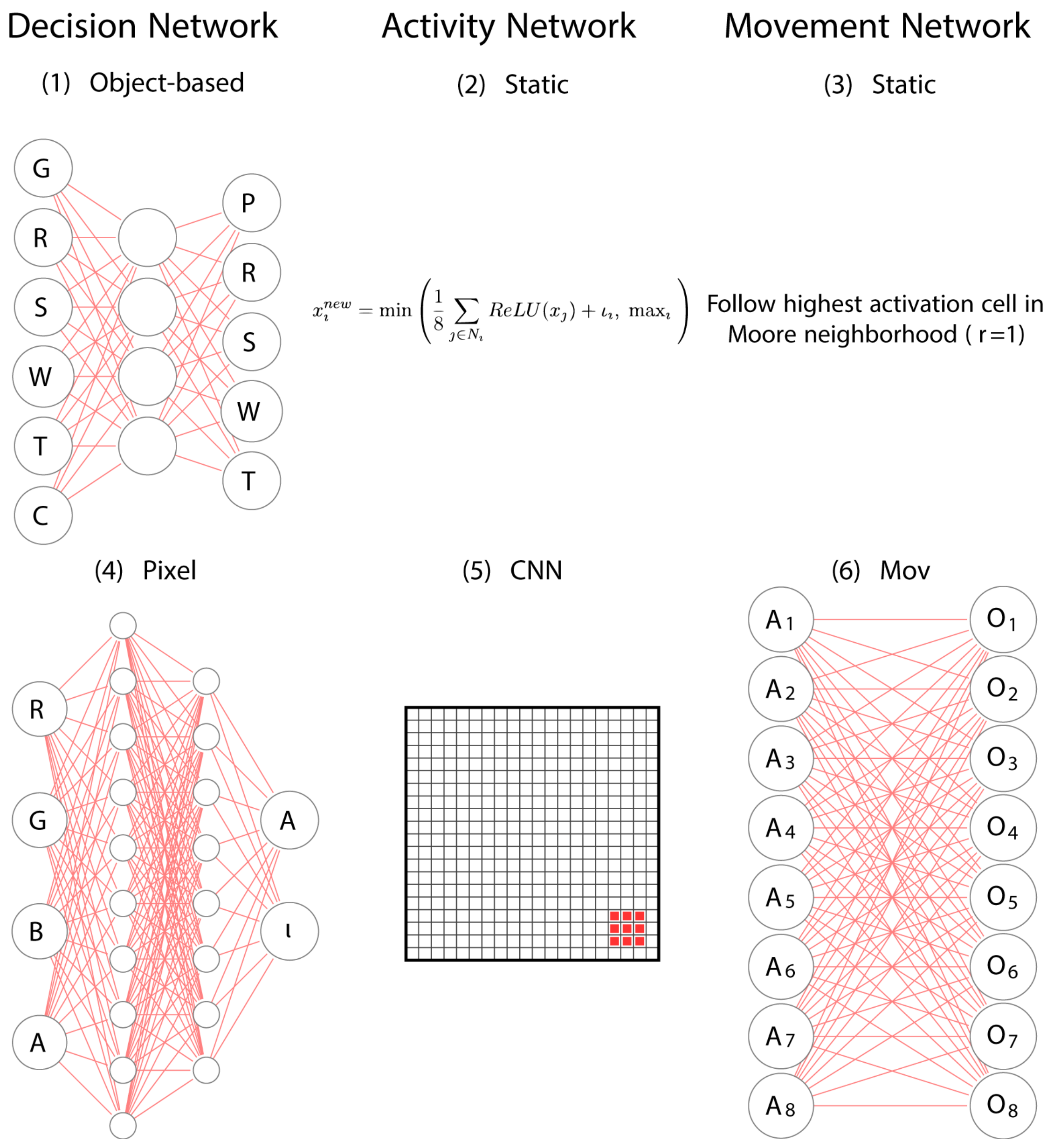

Decision Network
Activity Network
Movement Network
(1) Object-based
(2) Static
(3) Static
G
R
S
W
T
C
P
R
S
W
T
$x_\imath^{new} = \min\left(\frac{1}{8}\sum_{\jmath\in N_\imath} ReLU(x_\jmath) + \iota_\imath,\ \max_\imath\right)$
Follow highest activation cell in Moore neighborhood (r=1)
(4) Pixel
(5) CNN
(6) Mov
R
G
B
A
A
ι
A1
A2
A3
A4
A5
A6
A7
A8
O1
O2
O3
O4
O5
O6
O7
O8

Figure 6.1: **Neural architecture.** The Decision Network takes the agent's position **(1)** (g=grass, r=resource, s=stone, w=water, t=trap, c=carrying flag) or an object's colours (r=red, g=green, b=blue, a=action flag). The output are iota values for each object **(1,4)**. An array of iotas are passed to the activity network as a matrix input. **(2)** Uses the stated equation and **(5)** uses a 3x3 kernel to perform recurrent convolution on input, synchronously updating activities in the same layer; the network is activated a fixed number of times. The activity network's output constructs an activity landscape, a dynamic landscape used to determine agent's next position. The movement network uses the agent's neighbourhood values to produce an output. **(3)** uses the stated equation and **(6)** is a standard fully connected network. Links in red represent evolvable weights.

Each offspringś genome has a 20% mutation rate. Mutation involves a Gaussian noise value replacing a single parameter. This process continues until the remaining 90% are generated.

Decision Network	**Object**	As detailed in section 3.2.1
	Pixel	As detailed in section 6.2.1
Activity Network	**Static**	As detailed in section 3.2.2
	CNN	As detailed in section 6.2.2
Movement Network	**Static**	As detailed in section 3.2.2
	Mov	As detailed in section 6.2.3

Table 6.1: **Labelling for the sub-networks in the modular architecture**.

6.3 Results

Model	(0-4]	(1-4]	(2-4]	(3-4]	[4]
Pixel - Static - Static	100%	100%	100%	100%	100%
Pixel - CNN - Static	100%	100%	80%	80%	75%
Pixel - Static - Mov	100%	100%	100%	70%	60%
CNN Only	100%	100%	100%	100%	100%
Object - CNN - Static	100%	100%	100%	100%	100%
Pixel - CNN - Static	100%	100%	80%	80%	75%
Object - CNN - Mov	100%	100%	30%	25%	25%
Mov Only	100%	100%	100%	100%	100%
Pixel - Static - Mov	100%	100%	100%	75%	60%
Object - Static - Mov	100%	100%	100%	70%	60%
Object - CNN - Mov	100%	100%	30%	25%	25%
Object - Static - Static	100%	100%	100%	100%	100%
Pixel - CNN - Mov	100%	100%	15%	15%	10%

Table 6.2: **Distribution of final-generation fitnesses, taken over twenty runs for each of the ten neural network combinations.** Three combinations are shown twice each, for ease of comparison.

The experiments carried out were designed to provide direct comparisons between the new general-purpose (generalised) neural network designs and the previous domain-specific designs. Each of the three sub-networks has two designs (old domain-specific and new general-purpose), giving eight (2^3) combinations i.e. full network designs. For brevity we use the labels shown in table 6.1 for the six ($3 * 2$) sub-network designs. Each of the eight combinations (full network designs) is named according to its decision, activity, and movement sub-network designs. Object-Static-Static represents the previous domain-specific network, in which just the domain-specific Object Decision Network's weights are evolved. Pixel-CNN-Mov represents the full new general-purpose network, in which all three general-purpose sub-networks' weights are evolved.

In each of the eight combinations (full network designs) above, all non-static weights are evolved. In the Pixel-Static-Static full-network design, only the Pixel sub-networks weights are evolved as the other two sub-networks are static designs. To the eight combinations we add 'CNN Only', in which the full network uses a pre-evolved Object Decision Network and the Static Movement Network, such that only the CNN Activity Network weights are evolved; and 'Mov Only', in which the full network uses a pre-evolved Object Decision Network and the Static Activity Network such that only the Mov Movement Network weights are evolved.

Twenty runs were carried out for each of the ten combinations. Figure 6.2 shows fitness results over evolutionary time. Table 6.2 shows the distribution of final-generation fitnesses, taken over 20 runs for each of the ten neural network combinations.

After achieving maximum fitness (4), agents are evaluated in a Robustness Test. In this, each agent is evaluated in 10^4 deterministic RC world configurations (identical each run) with a river width of 3 (the most difficult width used during evolution). Performance on this test represents an agent's ability to perform in general RC environments, rather than just the 40 it was previously last evaluated in. Figure 6.3 shows completion rates in the Robustness Test, for each interchangeable network combination, with higher values demonstrating a greater quantity of robust agent controllers. Figure 6.4 shows agent age at completion in the Robustness Test, for each interchangeable network combination, with lower values demonstrating better quality (greater efficiency) in the behaviours of the agent controllers. These results are also covered in detail below, first for each sub-network individually and then for the full network.

6.3.1 Decision Network Results and Analysis

Other than the full Pixel-CNN-Mov network, all combinations with the Pixel Decision Network were able to achieve the highest fitness reliably, as shown in figure 6.2 and table 6.2. When compared to the Object Decision Network, equal robustness is seen when using the static activity and Movement Networks, shown in 6.3 but there is a sizeable drop in robustness when the CNN (recurrent ConvNet) Activity Network is used; indeed, the resulting robustness is lower than that in the full Pixel-CNN-Mov network, indicating that the static Movement Network is not well suited here and validating the approach of enabling an evolvable motion network. Finally, in all three comparisons, the Pixel Decision Network performed statistically worse than its Object counterpart in terms of efficiency, as shown in figure 6.4. Overall, the general-purpose Pixel Decision Network is still capable of long-term deliberative planning but with

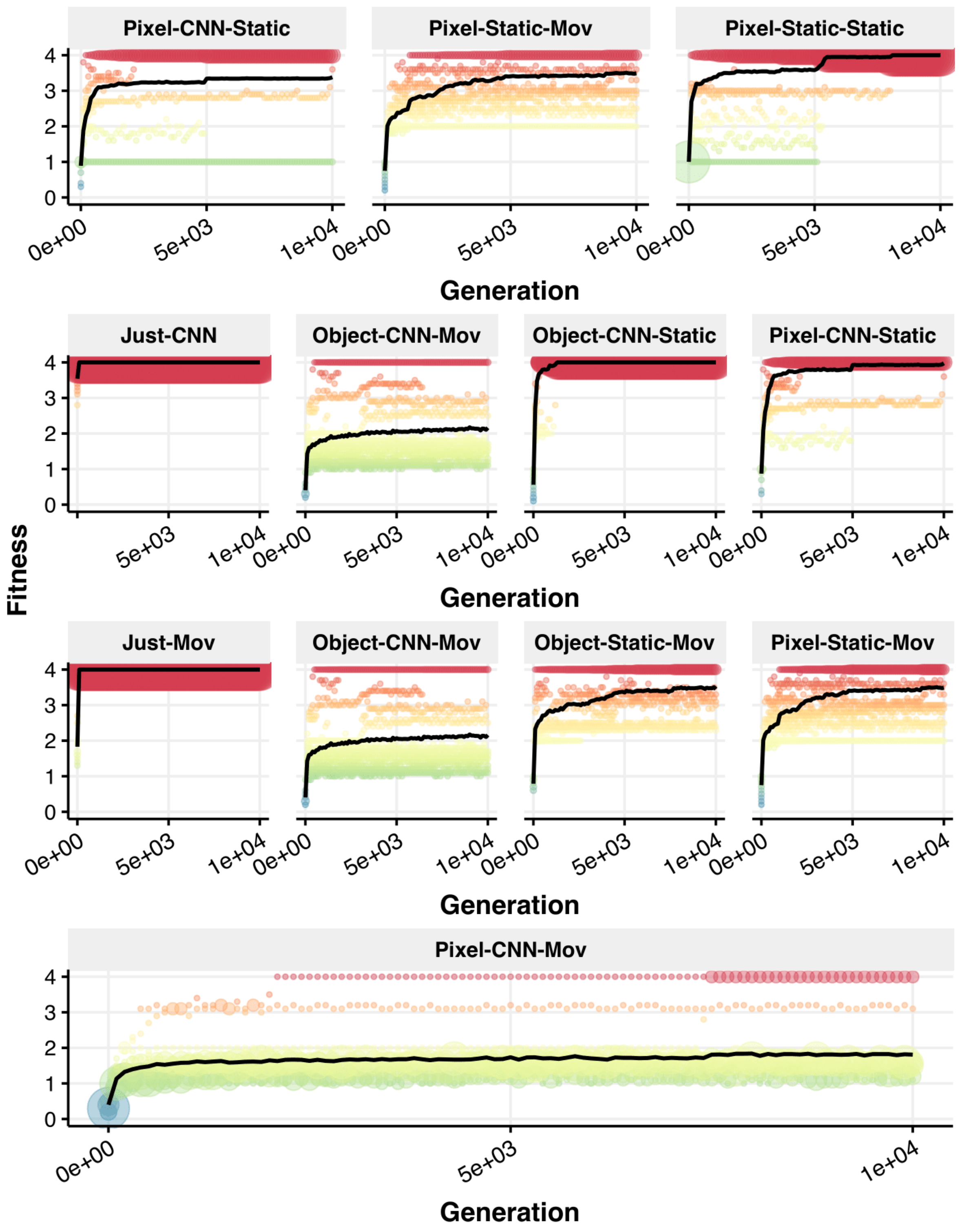


Figure 6.2: **Training data for each interchangeable network combination.** The solid line represents the mean fitness of the fittest individuals from 20 runs of the RC Task. Each dot represents an agent's fitness. The radius of the dot is increased when agents occupy the same fitness. Each combination is labelled above its corresponding graph.

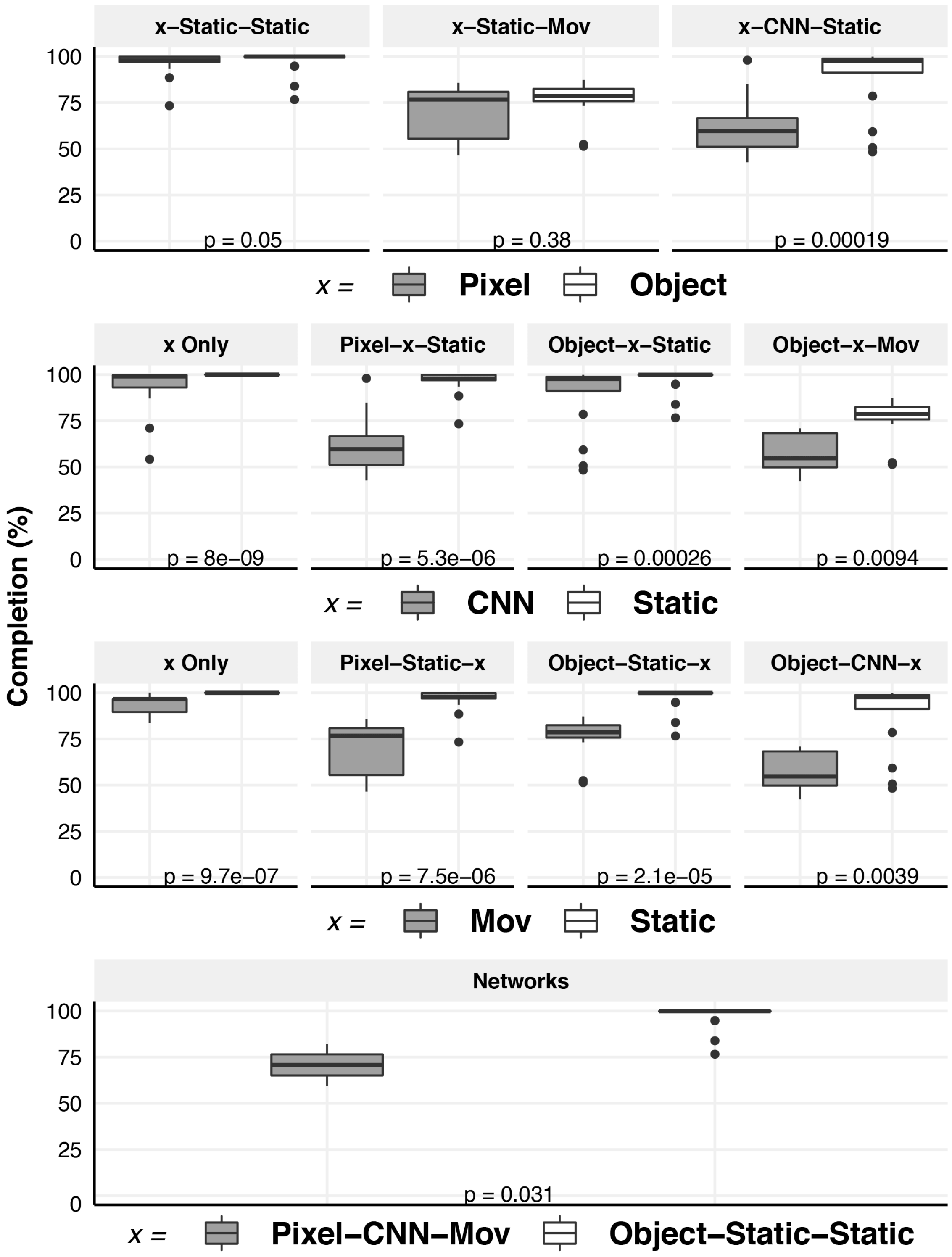


Figure 6.3: **Completion rates in the Robustness Test, for each interchangeable network combination.** In the Robustness Test, successfully evolved agents are evaluated in 10^4 RC world's at river width 3. Each pair or result sets was evaluated on the Mann-Whitney U test, with p-values given in each graph.

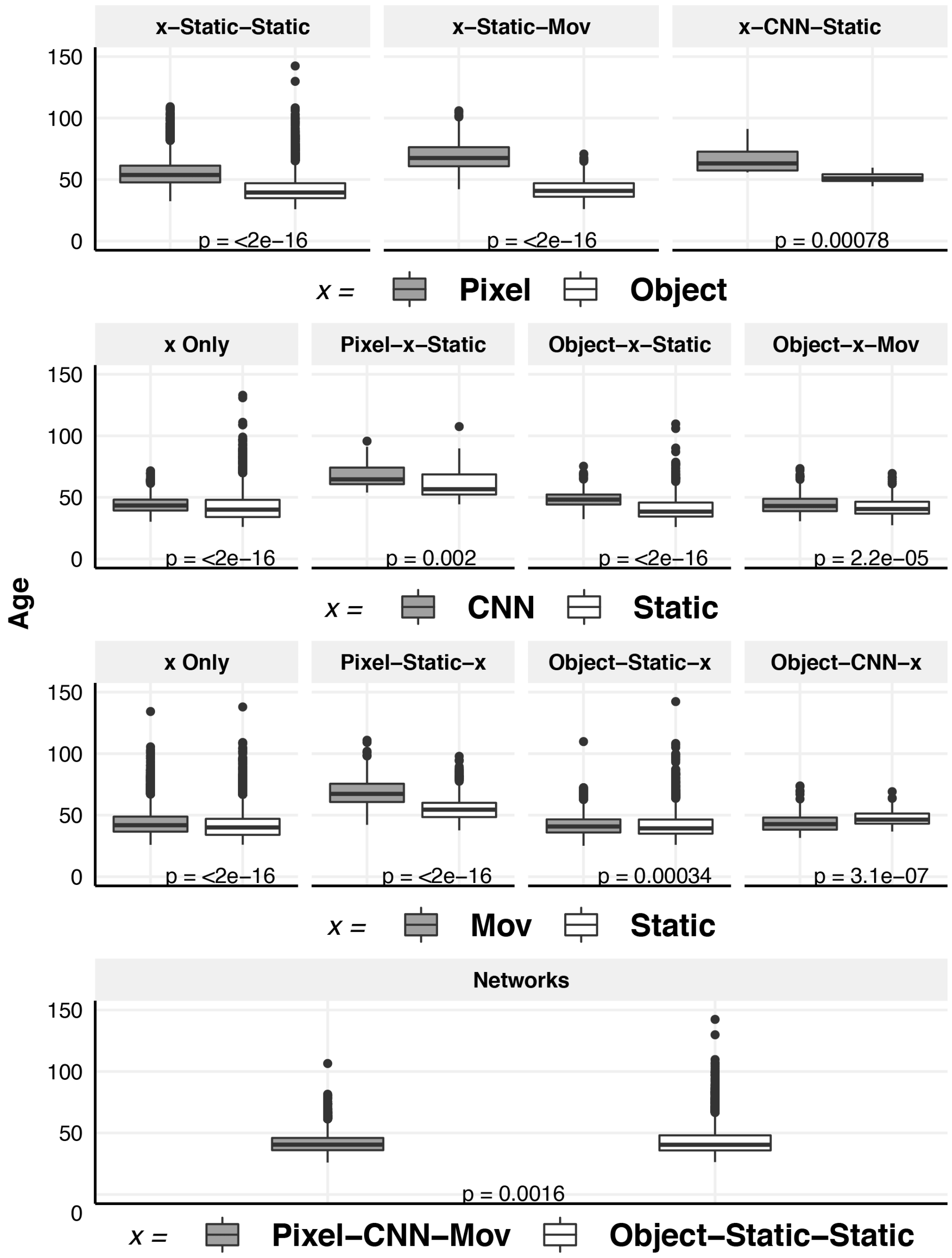


Figure 6.4: **Efficiency in the Robustness Test, for each interchangeable network combination.** In the Robustness Test, successfully evolved agents are evaluated in 10^4 RC world's at river width 3. Final ages are included only for those RC worlds (of the 10^4) in which all runs for both network designs were successful, allowing performance to be compared directly. Each pair or result sets was evaluated on the Mann-Whitney U test, with p-values given in each graph.

inferior robustness and efficiency results when compared to its domain-specific Object counterpart.

6.3.2 Activity Network Results and Analysis

All combinations with the CNN (recurrent ConvNet) Activity Network were able to achieve the highest level of fitness, as shown in figure 6.2 and table 6.2. However, when combined with the evolvable Movement Network, there is a sizeable drop in both fitness and robustness, with successful evolution dropping from 100% to 30% at RC worlds with river width two. This suggests that something about the task (locating stones, avoiding traps, bridge building, etc.) is difficult for evolvable motion planning when combined with evolvable movement. Combining CNN with the Pixel Decision Network also led to a sizeable drop in robustness. In terms of efficiency, the evolved robust networks containing the general-purpose CNN Activity Network were a little less efficient than those with the Static counterpart on average (median) but resulted in fewer inefficient outliers, a potentially important benefit.

6.3.3 Movement Network Results and Analysis

All combinations with the evolvable Movement Network were able to achieve the highest level of fitness, as shown in figure 6.2 and table 6.2. When static motion planning (Activity Network) is used (with either Decision Network design) 100% success is achieved up to river width 1 (fitness 2.0); past that performance drops. To achieve fitness 3.0, agents must build a connecting bridge design by locating and correctly moving multiple stones. To achieve fitness 4.0, agents must be able to do this same behaviour consistently, which the current Movement Network struggles with. No combination without the evolvable Movement Network exhibits this problem to this extent. It may be that a deeper Movement Network (with at least one hidden layer) would perform better. All evolvable Movement Network combinations also have statistically inferior robustness. The combinations with Static movement have a relatively consistent median of 97%-100% robustness, whereas the medians for the evolvable-movement combinations are between 55% and 88%. Object-CNN-Mov provided statistical superior efficiency over its counterpart; this again gives credence to the ability to achieve more efficient movement when motion planning and movement are evolved together. To conclude, the evolvable Movement Network has a sizeable negative effect on robustness but can result in more efficient movement when motion planning and movement are evolved together.

6.3.4 Full Network Results and Analysis

When all general-purpose evolvable networks are used, this architecture still retains long-term deliberative planning, as shown in table 6.2. However, this combination produces the largest drop in agents achieving maximum fitness. Figure 6.2 shows that a majority of the agents stay between fitness 1-2. So, agents are successful at locating the resource but struggle to consistently locate stones and place them on the single-cell river. The limited proportion of successful agents could, in turn, be responsible for the relatively high robustness score.

While a number of the incomplete general-purpose networks have medians with robustness in the 50%-60% range of completion, the full network produces a consistent 73% median completion rate on the 10^4 RC Worlds. In terms of efficiency, the full general-purpose network sees a statistical advantage over the domain-specific full network, with the age of the agents being very slightly (0.05) lower than the original and, potentially important, there being fewer inefficient outliers. Overall, these results show that maximum fitness, robustness, and efficiency can be achieved without domain-specific designs.

6.3.5 Further Analysis

This section is to achieve an understanding of the questions raised during results gathering. Firstly, fitness is an aggregation of 10 RC runs. To achieve fitness 4.0 agents would have to successfully complete 40 RC worlds. So, why might agents fail significantly in the Robustness Test despite being able to achieve high fitness in training. Figure 6.5 offers some insight, it shows the distribution of deaths in relation to agents' robustness scores. The graph suggests highly robust agents primarily avoid harmful objects and fail by not reaching the resource in the allocated time. This is seen in the rise of age deaths and the fall in traps and water deaths when robustness reaches 100%. Age is when the agent is unable to obtain the resource within 150 time steps. Traps always kill and water will only kill if the agent does not place a stone down on that cell. As robustness lowers to 80%, age deaths decline; traps increase to represent half of all deaths; water matches the distribution of age. Below 80% robustness, deaths are gen-

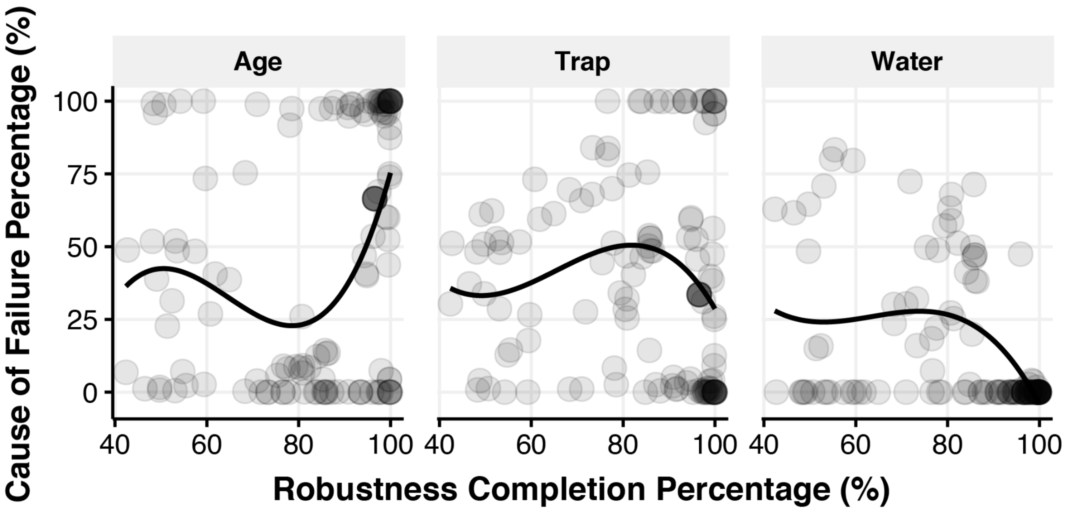


Figure 6.5: **Scatterplot showing the relationship between the cause of agents deaths and the completion achieved during a Robustness Test.** The line represents the conditional mean of the two data points.

erally equally distributed with no inherent cause being the dominant factor. Together these results provide important insights into desirable strategies. Those highly robust strategies are those which fundamentally avoid harmful objects. Water is a necessary interaction for bridge building but traps are not. A possible reason for such a higher distribution of trap deaths is training not accounting for the negative traits of traps. Hence why the distribution of trap deaths rises as robustness is lowered. Understanding this brings some insight in Figure 6.6; this isolates the cause of death distributions to each network combination.

An interesting observation is the rise of water deaths when using the evolvable Movement Network. To compare, Static movement sees agents with very low number of deaths due to water (less than 5%) or none at all. Due to our understanding that water deaths are rare among highly robust individuals, this offers an explanation

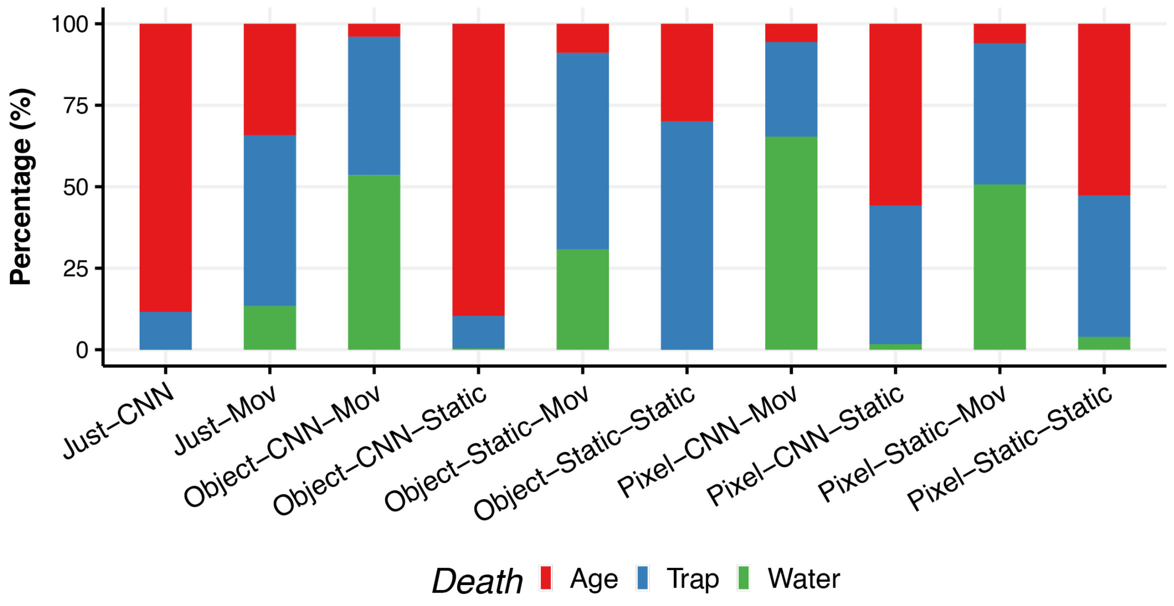


Figure 6.6: **Distribution of deaths from the fittest agents through 10^4 RC worlds seen during the Robustness Test.** All network combinations are shown. Age deaths are agents who were unable to find the resource under 150 time steps. Trap deaths are caused by entering into trap objects. Water deaths are caused by entering into water objects without placing a stone.

into the evolvable movement's influence on robustness. Yet, the evolvable Movement Network can replicate the static behaviour but appears not to. To analyse further, each successful evolvable Movement Network combination was simulated again on the Robustness Test, but the evolvable Movement Network was replaced for the static behaviours. It is important to note, these agents were not trained to operate with the static movement behaviours previously.

Figure 6.7 shows two graphs. The top section compares the robustness of the two sets of data. The bottom shows if any of the 10^4 worlds both produce the exact same movement. Interestingly, the evolvable Movement Network was able to find strategies outside of static rules despite all other networks being pre-evolved to that ruleset. This network restriction would theoretically lead the evolvable Movement Network to the same behaviour as a static movement. Yet, under half the runs showed new movement strategies when presented with the same RC world. Turning now to the evolvable network combinations. In Pixel-Static-Mov and Object-CNN-Mov, both their counterparts produced statistically superior robustness. This is interesting as both prosper with behaviour they were not trained for. Object-Static-Static produces a higher median but a large range; yet statistically indistinguishable. Overall, despite the static movement behaviour being a superior strategy evolution does not replicate it during training.

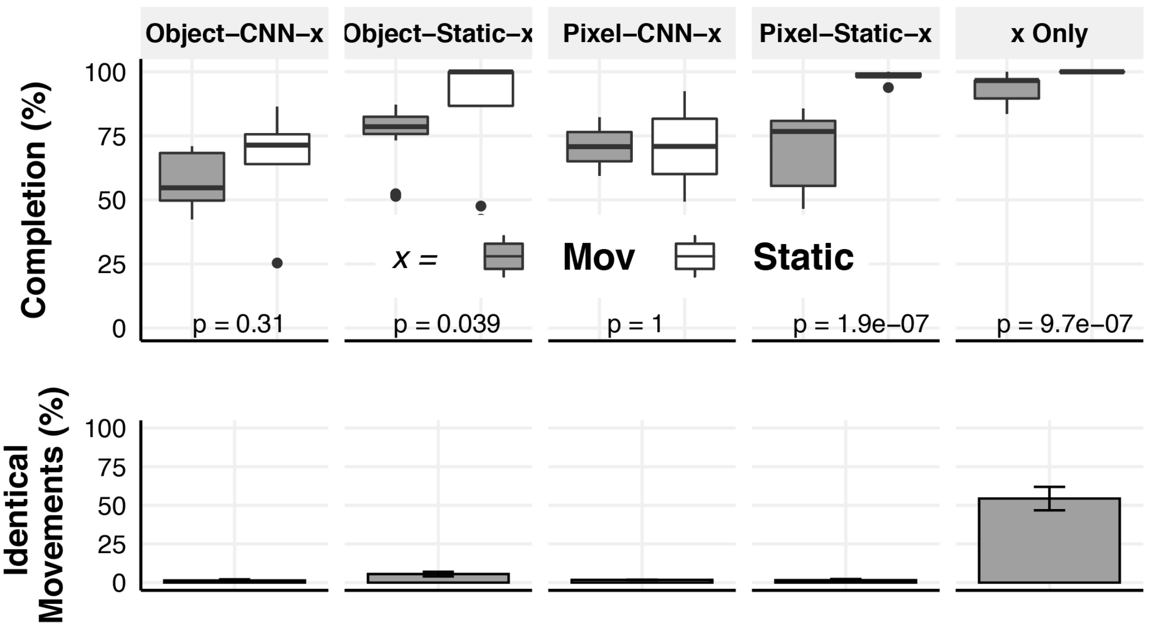


Figure 6.7: **Completion rates in the Robustness Test, for each interchangeable network combination that has been successfully trained with the evolvable Movement Network and replacing those with the static Movement Network.** In the Robustness Test, successfully evolved agents are evaluated in 10^4 RC world's at river width 3. Each pair of result sets were evaluated on the Mann-Whitney U test, with p-values given in each graph.

6.4 Discussion

A goal of this chapter was to identify if the modular architecture could maintain success with the removal of domain specific features. By achieving fitness 4.0 in the RC Task with the full network architecture (Pixel-CNN-Mov) in multiple runs this proves it is possible. Achieved with a simple GA with a standard mutation rate, it is reasonable to assume tweaks to the evolutionary process could improve training performance. The next sections will delve into each generalised network's performance.

6.4.1 Successes and Failures Decision Network

The success of a Pixel representation achieving the highest fitness in all network combinations, suggests two restrictions can be removed from the previous architecture. The first may be the clearest, the removal of binary inputs. This prevents the Decision Network from having domain specific knowledge before run time. Secondly, allowing continuous outputs, and therefore allowing hierarchy in item desirability. The Object Decision Network's output iota defaults to three states; undesirable (-1), neutral (0) and desirable (1). Instead, the Pixel's output uses a continuous number from -1 to 1. This allows nuance in the importance of items. This feature does not benefit this particular task as only one item is desirable at a time (If an agent is not holding a stone, a stone is desirable; if an agent is holding a stone, the river is desirable). But, there is potential for tasks where many items have different levels of importance at one time. Despite the positive result, the quality of the Pixel representation's solutions is less than the object counterpart in all aspects; especially when used with the recurrent ConvNet.

6.4.2 Successes and Failures Activity Network

Recurrent ConvNet achieving success in all combinations is of great significance. By replacing static behaviours with an evolutionary process, it allows future implementations to remove the static constraint. Especially considering the results of Object-CNN-Static which saw all runs reaching the highest fitness while achieving robustness close to the original. In comparison to HyperNEAT, the indirect motion planning method provided some success; but, had restrictions and was only capable while behaviours were pre-evolved and static [107]. To conclude, the use of a recurrent ConvNet does produce a feasible motion planning network and a replacement for the static shunting model. In the future, it may be worth investigating Leaky ReLU. This would allow negative values from the Decision Network to have a greater significance of a region of local areas, rather than a specific cell.

6.4.3 Successes and Failures Movement Network

The evolvable Movement Network's biggest potential criticism is the formatting of inputs. The inputs take a binary activation of the highest surrounding activity value. This removes the responsibility from the Movement Network to discover the behaviour through training. This was a purposeful choice as the responsibility of activity values is already held by the Activity Network. It seems unnecessary for the evolvable Movement

Network to learn this behaviour also. Yet, this could bias the results to achieve the same behaviour as the static Movement Network; which in turn, would remain a domain specific network. However, as we have seen in figure 6.7 this does not happen. Instead, less than 10% of 10^4 runs saw identical movement between the static and evolvable Movement Network, when evolved with additional networks. Even when all networks are pre-evolved, the evolvable Movement Network still found novel ways to use the input. This assures that this network configuration can exhibit different behaviours to the static approach. This is especially true with the CNN (Recurrent ConvNet) where together the efficiency results rival their static counterparts.

The inclusion of evolvable movement saw the largest impact on training and robustness. Further analysis suggests the reduction in quality is due to the behaviours the Movement Network adopts; in which traps and water deaths become more common. Theoretically, the evolvable Movement Network is provided all the information to avoid both traps and water. If trained correctly, the Decision Network produces negative activity values for the corresponding cells. When paired with the static motion planning, water, and traps will always represent the lowest surrounding cell. But as shown, the Movement Network rarely adopts the strategy to follow the highest activation. So, the model may benefit from additional information to aid in identifying objects to avoid. Such as, a negative binary value to state the lowest activation, and the inclusion of hidden layers.

6.5 Deep Neuroevolution Comparison

To further emphasise the results of the previous section, a direct comparison will be made with another general architecture that has achieved state-of-the-art results in ALE; deep neuroevolution with deep ConvNets. Currently, deep ConvNets play a vital role in achieving state-of-the-art results across a broad array of machine learning domains. Including; computer vision [105, 228], robotics [111, 147], self-driving cars [17], sound [242, 6, 174], art [85, 139] and others. These architectures have historically been utilised with Reinforcement learning (RL), but recently it has been shown that Neuroevolution (NE) can achieve comparable performance; setting its own state-of-the-art results [226, 256]. Yet, each of these techniques has suffered in environments which require long-term planning. Pitfall and Montezuma's Revenge are the most researched of these types of environments. Very recently, Go-Explore established an RL algorithm which performs well on these tasks [57]. Yet, NE's leading method is yet to complete either challenge.

Instead of focusing on these complex examples (Montezuma's Revenge and Pitfall), the RC Task acts as a similar benchmark; this is due to the similarity in the task domain with reduced complexity. The fundamental objective of Montezuma's Revenge and Pitfall is to traverse multiple scenes before receiving rewards, all while avoiding lethal objects. For example, the first scene of Montezuma's Revenge requires the agent to reach the bottom of the scene; traverse a ladder; collect a key; traverse to the opposite side of the screen; unlock the door; while avoiding a lethal moving object.

Agents will also fail if they fall from a great height. Collecting the key and unlocking the door provides a reward of 100 and 300, respectively. Despite achieving a reward for collecting the key many approaches are unable to proceed past the reward score 0 on the task, including; deep neuroevolution [226] and DQN [14]. The RC Task is simplified in three ways; all information necessary to complete the task is available in one screen; the task is incremental and focuses agents on specific behaviours at a time; there are only four objects in the environment. The analysis of deep neuroevolution on the RC Task is to assess whether simplifying the domain allows evolution to overcome issues associated with sparse reward tasks which require long term deliberative planning. As the previous section showed a NE general-purpose model which can overcome the issue of sparse rewards.

6.5.1 Convolutional Neural Architecture

A comparable experiment setup is required to compare deep neuroevolution's performance to the Pixel-CNN-Mov (section 6.3); Pixel-CNN-Mov being the architecture that removed the greatest number of constraints in the previous section. Firstly, there is a common architecture for ALE used across RL and NE which was first established with Deep Q Networks (DQN); the RL algorithm utilising deep Q-learning [157]. As DQN was the first algorithm to achieve human-level control in a large fraction of Atari 2600 games, the topology has become a standard to compare various algorithms to one another. As this architecture has been successful in a majority of games, across different learning strategies, it will be used here on the RC Task; however, some changes will be necessary. The original ALE DQN architecture comprises of; 84x84x4 tensor input; three convolutional layers; two fully connected layers; with a single output for each valid action. DQN pre-processes the ALE pixel input to greyscale and downsamples the last 4 frames to an 84x84x4 tensor. Whereas, the Pixel decision network uses colours as inputs. For a fair comparison, both greyscale and RGB are used. The RC Task's input would be 20x20, which is already significantly lower than the down-scaled ALE screen input. So, the input size will remain the same but the remaining architecture may prove too large for a reduced input. So, two architectures are used. First, the original DQN architecture with a 20x20 adapted input. Second, a smaller DQN is provided in the deep neuroevolution code. This reduces; size of layers; kernel sizes; the number of filters and strides; Table 6.3 shows the specifics. There are 18 outputs, as there usually are with ALE games. Half of the outputs represent a movement position in the agent's Moore neighbourhood. Then, the next half is the same positional movements, but with the addition of the agent's 'action'; in this domain, it will be agent's ability to pick up and put down items. The output with the highest activation will dictate to the agent where to move and what action to do.

	Large-DeepNeuro				**Small-DeepNeuro**			
Layer	**Size**	**Filters**	**Kernel**	**Stride**	**Size**	**Filters**	**Kernel**	**Stride**
Conv #1	-	32	8	4	-	16	8	4
Conv #2	-	64	4	2	-	32	4	2
Conv #3	-	64	3	1	-	-	-	-
FC	512	-	-	-	256	-	-	-

Table 6.3: **Topology specifications for ConvNet used on the RC Task with deep neuroevolution.** (FC = Flattened Layer, Conv = Convolution Layer)

Next, training times for ALE are reported in frames processed and time steps but, for an apt comparison, this work will show results in the number of generations. Populations size and elitism remain the same at 1000 and 10%. The mutation rate is 0.002, which remains the same as the ALE work in deep neuroevolution.

Finally, The RC Task will incorporate a visual change for when an agent is holding a stone. Previously, this information was not a visual component and passed to the network as a boolean variable. In this type of topology, there is not an obvious way to incorporate this. Instead, when observing ALE games, and the networks which successfully play them, games incorporate visual changes to indicate a change in domain state. Alien and Ms Pac-Man change the colour of enemies when they can be consumed; Amidar changes the colour of paths that agents traverse and rectangular portions of the board are filled when a connecting path has been achieved; Frostbite sees pieces of ice change from white to blue when the agent jumps on them, all need to be jumped on to progress. Therefore, the agent will change colour when holding a stone.

In ALE and the hard maze task, deep neuroevolution included the three previous frames in the input to provide temporal information. In the RC Task there are only 400 discrete positions an agent can traverse to. Atari games were designed as a visual experience, so agents may take many frames to reach a discrete position or agents have greater discrete positions; this provides the user with a sense of motion. However, the RC Task does not provide frame intervals between discrete positions. So, it is a question whether this should be included. Although not a direct comparison, AlphaGo Zero also implements past frames of Go as a history of players moves [209]; Go is a board game of limited discrete positions. In order to provide the RC Task with adequate information, the previous three frames will also be included.

6.5.2 Results

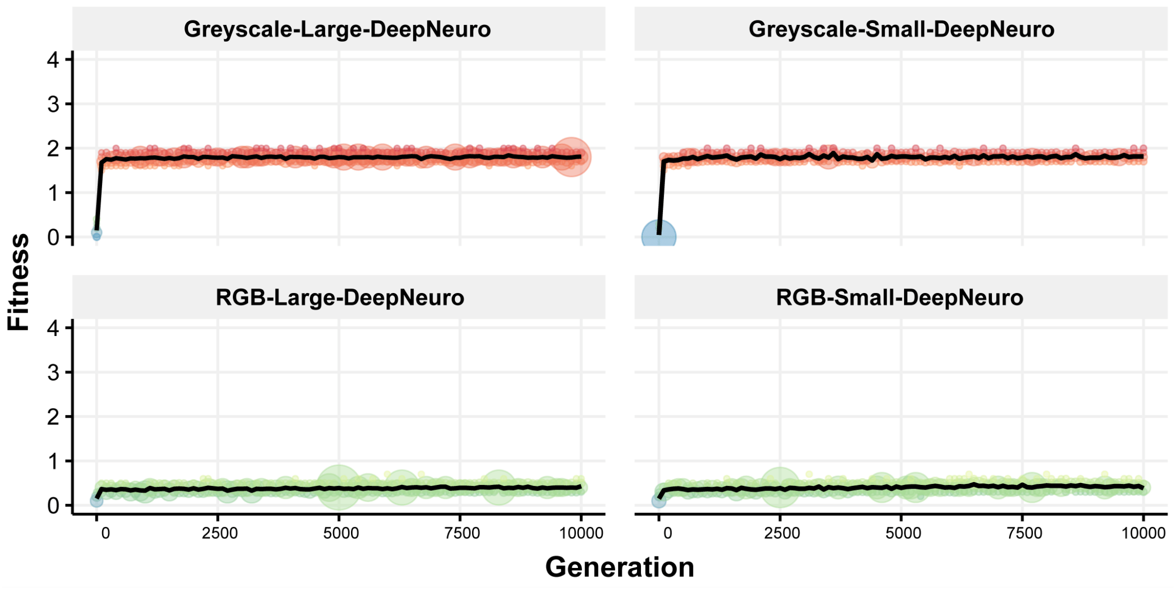


Figure 6.8: **Training data for all network combinations for deep neuroevolution on the RC Task.** Solid line represents the mean fitness of the fittest individuals from 20 runs of the RC Task. Each dot represents an agents fitness. Radius of dot is increased when agents occupy the same fitness. Each network combination is labelled above its corresponding graph.

As figure 6.8 shows, no approach achieves long term deliberative planning in the RC Task. The most successful agents achieved fitness 2.0; attributed to the greyscale input. As previous results have shown, fitness beyond 2.0 is difficult due to the behaviours required. RGB inputs were detrimental to the performance in both network sizes; with none achieving higher than fitness 1.0. Greyscale appears superior in two ways, (1) the input dimensions are reduced to 1/3 and (2) the different scales of luminance per colour channel still allows objects to be distinguishable. Despite the larger size, Large-DeepNeuro performed just as well as Small-DeepNeuro; both inputs exhibited seemingly identical performance. A visual of the large ConvNet is shown in figure 6.9 for greater comprehension of the network.

6.5.3 Discussion

This section interprets why deep neuroevolution could not achieve long term deliberative planning in this domain. The first and second RC worlds require fairly rudimentary behaviours to complete. The first RC world requires agents to locate a deterministic resource location. As this resource position is static the behaviour becomes simple to converge once achieved. The second RC world requires stones to be placed on the water to create a 1 cell bridge. Stones have stochastic placement so agents can not simply remember a specific motion pattern to retrieve them. But, stones have a chance to be aimlessly picked up and put down while traversing to the resource; then, this behaviour remains in the population by receiving a reward and surviving the selection process. As the population is heavily focused around fitness 2.0 it shows they are achieving this behaviour consistently and not by chance.

RC worlds three and four require agents to go back and forth between stone and river. As stated, previous necessary behaviours could be stumbled upon and reinforced via the population converging on that area in the search space. But now, traversing back to a stone requires greater deliberate behaviour which is difficult to discover randomly; especially since the population remains in a search space area that establishes moving forward to the resource is desirable behaviour. The fitness function does not

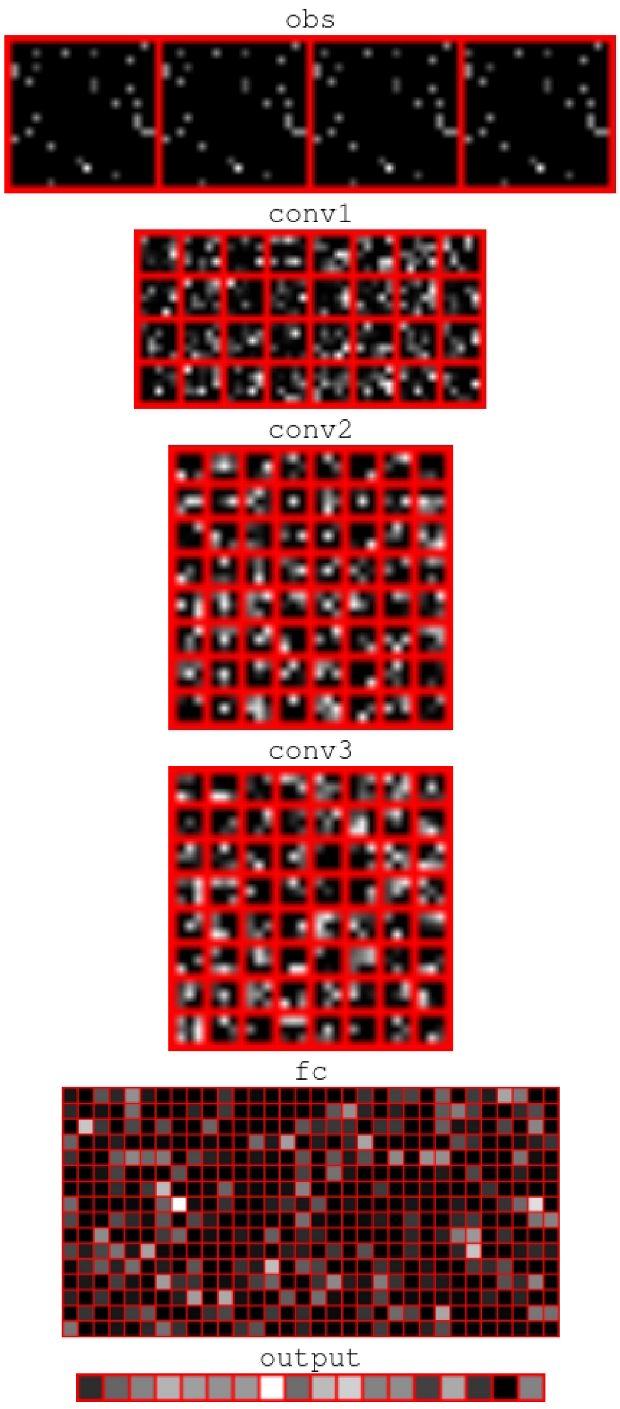


Figure 6.9: **Visualisation of the (large) convolution network for River Crossing Task (RC Task) in greyscale representation**. From top to bottom on the right are: the processed observations, and then the activations for the convolutional layers, the fully connected layer, and finally, the movement outputs. Brighter activation indicates higher value. Atari Zoo was adapted for this visualisation. [227]

provide granular feedback, such as a negative reward for dying via water or traps or that putting stones on the river is positive. As a result, there is no mechanism for the population to diversify greatly from the already established positive behaviours of RC worlds one and two. With the architecture laid out in section 6.2, sub-goals are active within the architecture during this problem. The change in Decision Network will produce a dramatically different Activity landscape, which in turn causes the Movement Network to produce new behaviours. But, ConvNets see little change in their input activations to expect a dramatic change in behaviours after a stone is placed on the river; shown in figure 6.9.

A suggestion could be the inclusion of diversity mechanisms into the evolutionary process such as; speciation, fitness sharing, crowding. This may provide an avenue for agents to uncover different behaviours for picking up stones and reaching the resource; one of which is beneficial at later stages. However, this could also be said for the architecture in chapter 6. A diversity method that should be avoided is a change in the fitness function, such as; novelty search, multi-objective evolutionary algorithms; quality diversity, etc. This then focuses the challenge away from sparse rewards. For example, a common use case for novelty search is a hard maze task in which objective fitness becomes trapped at local optima when trying to solve the maze. But, when the reward is changed to finding novel positions, each run, the maze is able to be solved. So, a novelty search method for the RC Task could be devised which focuses on the placement of stones and creating unique placements. This can then be extended in works like quality diversity which sees different types of specific bridges being created (widest, longest, most stones used, etc). This would inevitably lead to a search space in which a solution to solving the RC Task, with a 3 stone connecting bridge, is much more achievable. But, in the process, the task would change into something much simpler to exploit.

Additionally, this task is not inherently deceptive which these methods are usually targeted towards. Rewards are not luring agents into a local optimum to increase the objective reward. Both traversals to the resource and picking up stones are both behaviours that agents need to progress. It is the lack of discovery of new behaviours which plateaus the model's performance.

6.6 Conclusion

The contributions of this chapter are: (1) Further validating the use of a recurrent ConvNet, which distributes activations across neighbouring neurons to create pathways for motion planning. In all network combinations, the ConvNet still achieved the most difficult behaviours required. And, when paired with an evolvable movement network, efficiency could increase over the static implementations. (2) Demonstrated the effectiveness of a general-purpose, modular, hierarchical architecture for long-term planning in domains with sparse rewards. The network contained no domain-specific aspects and although became a more difficult task for evolution, retained a high amount of robustness in stochastic environments. (3) Demonstrated an issue with the current state-of-the-art neuroevolution game player on a sparse reward task focusing on long-term deliberative planning. Deep neuroevolution becomes trapped at a local optima during training and is not able to produce the same behaviours the general-purpose, modular architecture is able to. For future work, it should be investigated whether aspects of the generalised modular network could be adapted to more proven techniques. This work limited the recurrent ConvNet to a specialised role. However, the distribution of high activating neurons to its neighbours could provide supplementary information during activation; especially in environments where the variation of input data at each activation is minimal. This will be investigated in the next chapter where the recurrent ConvNet is paired with a conventional ConvNet architecture and trained with deep neuroevolution.

7 | Applying Recurrent Convolution with Deep Neuroevolution

7.1 Introduction

With the emergence of deep neuroevolution (section 2.2.7) opportunities have emerged using GAs in domains that have largely been subjected to experimentation by RL; due to GAs achieving state-of-the-art performance with large ConvNets [226]. These results were achieved with a simple GA that only employed elitism and mutation, and not utilising established neuroevolutionary techniques such as crossover, exploiting regularities and a range of diversity mechanisms (section 2.3). This allows ample future exploitation in evolutionary experiments and a wide range of RL domains still to explore. So, deep neuroevolution offers a promising foundation and therefore, this chapter will start experimentation with deep neuroevolution as it is of greater benefit to the research community. The novel architectural contribution of this thesis has been the recurrent ConvNet and its ability to produce robust motion planning. Now, this chapter will assess the possibility of using the same recurrent ConvNet as part of a much larger conventional ConvNet architecture.

A theme seen in ConvNet and deep neuroevolution is taking established, working, techniques and, with advances in computation power, scale them to greater challenges; ConvNets were proposed in 1990 before the major success of AlexNet [121] in 2012 and GAs have been around since the 1970s but have not been scaled to 4M parameters until deep neuroevolution in 2018 [5]. The previous chapters have shown how well a recurrent ConvNet has worked on a single 20x20 input with a single 3x3 kernel for producing motion planning landscapes that, with simple rules, could produce long term deliberative planning. Although conventional ConvNets could not use the recurrent layer for motion planning, the resulting feature is still unique and would be difficult to create in a conventional convolutional layer. So, the question being assessed is does the recurrent layer produce benefits that a conventional ConvNet can not. This research question will be investigated on the multi-environment platform, the Arcade Learning Environment (ALE). ALE is shared with many RL algorithms as well as evolutionary ones, allowing a comparison not just with a GA but many different learning approaches.

7.2 Experiment Setup

Experiments use the open source code for deep neuroevolution[1], with additional methods to achieve recurrent convolution with Tensorflow. The following presents the details and rationale of the recurrent model. The evolutionary process evaluates each individual in the population each generation, producing a fitness score for each. The top 20% of the population become parents for the next generation and the fittest individual is

[1]https://eng.uber.com/accelerated-neuroevolution/

copied to the next generation unmodified. No crossover is used in deep neuroevolution, just mutation. So, a parent is selected uniformly at random and is mutated by applying additive Gaussian noise to the genome; this continues until the new population is generated. The new population is then evaluated and the process repeats for a defined number of generations or until some other stopping criterion is met. For ALE, the stopping criterion is evaluating 4e9 frames.

7.2.1 Neural Architecture

The implementation of recurrent links to a conventional ConvNet could be done in a number of ways. In the previous chapters, the recurrent links acted as a single kernel on a single feature. The obvious equivalent to that is applying recurrent links at the input layer. However in the previous works, recurrent convolution is applied after objects in the environment have been considered desirable or undesirable. A similar process takes place at the first convolution layer, where kernels extract specific features in which to retain. So, adding recurrency at this layer should mimic the process of previous implementations, as can be seen in figure 7.1.

This implementation of recurrent links requires an iteration parameter; this parameter dictates how often convolution is applied to the same layer. At each iteration, the original values at iteration 0 are incorporated back into the iteration x output via an identity kernel passthrough. Previously, this value would be chosen by the experimenter, but in this work the size of a feature is dynamic depending on the previous layer. This would make it difficult to have a parameter that fits a variety of network sizes. Therefore, the chosen iteration parameter is the width of the feature. A characteristic that has been important for this recurrent ConvNet to work previously, is the ability for an activation value from one corner of the feature to reach the corresponding corner. The iteration value being the width of the feature allows this to take place, it is then evolution's role to find the appropriate weights to not saturate the feature space.

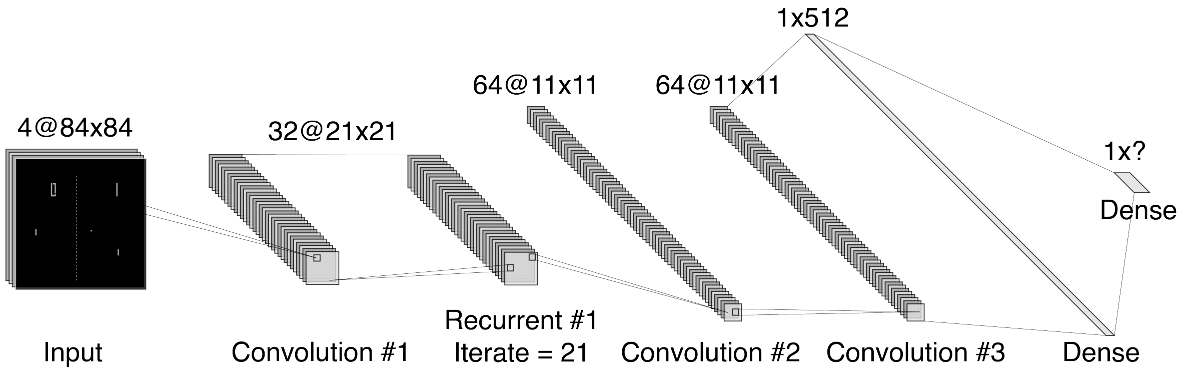


Figure 7.1: **Illustration of the recurrent convolutional neural network.** The input to the neural network consists of an 84x84x4 image, followed by one convolutional layer, one recurrent layer, then two convolutional layers and two fully connected layers with a single output for each valid action. The recurrent layer provides a 3x3 kernel per feature and applies convolution (as a convolutional neural network) then the new feature is combined with the original feature. This is an iterative process and is repeated until a certain number of times (for this work 21).

7.2.2 ALE Environments

Bellemare et al. [12], and later clarified in Ostrovski et al. [173], created a taxonomy of ALE games for agent behaviours and environmental rewards. Instead of simulating the recurrent ConvNet on each game, three are taken from each category in an attempt to see an overview of performance. This was necessary to overcome a resource restriction on the researcher's part. The original deep neuroevolution code was simulated on 720 cores for 6 to 24 hours to achieve 6 billion frames for each run. The code was later updated to allow pipe-lined CPU and GPU commands to speed up the code base. However, with 10 cores of an Intel Xeon E5-2680 with 2 Nividia 1080 GPUs runs could still range from 5-7 days with 6 billion frames simulated, on the original ConvNet architecture. By choosing games from each category this work aims to give a perception of performance across different domains, without testing every domain. Frames are also limited to 4 billion as it appears from the data in Such et al. [226] that after this point performance gains are minimal.

- 'Human-Optimal' refers to games in which agents have achieved human-level, or higher, performance but demonstrated behaviours as a human would.
- 'Score Exploit' refers to games in which agents have achieved human-level, or higher, performance without demonstrating behaviours as a human would.
- 'Sparse' and 'Dense' rewards are qualitative descriptors of the game's reward structure.

7.3 Results

Table 7.2 shows the final score of the elite (fittest) individual of each set of 5 runs. The elite's score is calculated via 30 independent episodes, using the same genome, and provides the mean; this is to account for the robustness of a given solution. As

Easy Exploration		Hard Exploration	
Human-Optimal	**Score Exploit**	**Dense Reward**	**Sparse Reward**
Asteroids	Kangaroo	Alien	Solaris
Crazy Climber	Seaquest	Amidar	Private Eye
Skiing	Krull	Frostbite	Venture

Table 7.1: **Atari 2600 games and their taxonomy selected for these experiments.** The taxonomy is based on the games exploration difficulty found in Ostrovski et al. [173].

Frames	Standard 4B	Recurrent 4B	Difference (%)	p Value
Asteroids	2120	3690	54 (+)	**2.506e-07**
Crazy Climber	68600	84600	21 (+)	**6.255e-05**
Skiing	-5540	-5540	0	1.0
Kangaroo	11300	14500	24 (+)	**2.2e-16**
Seaquest	1500	1760	16 (+)	**.03286**
Krull	10900	9500	13 (-)	**.0005632**
Alien	3070	4090	29 (+)	**2.2e-16**
Amidar	418	462	10 (+)	**2.2e-16**
Frostbite	8640	8510	2 (-)	**6.704e-14**
Solaris	5460	5360	2 (-)	.371
Private Eye	15200	15100	1 (-)	**3.978e-05**
Venture	1410	1340	5 (-)	**0.00614**

Table 7.2: **Results on 12 Atari games with a conventional and recurrent convolutional networks with a simple genetic algorithm.** Results represent the mean score of the elite (fittest) seed over 30 no-op independent episodes. The scores shown are the highest achieved over the 5 trained runs for 4 billion frames. The p values are taken from a two sample t-tests between the 30 scores achieved in the no-op independent episodes by the elite seeds. Those in bold are statistically significant (p < 0.05).

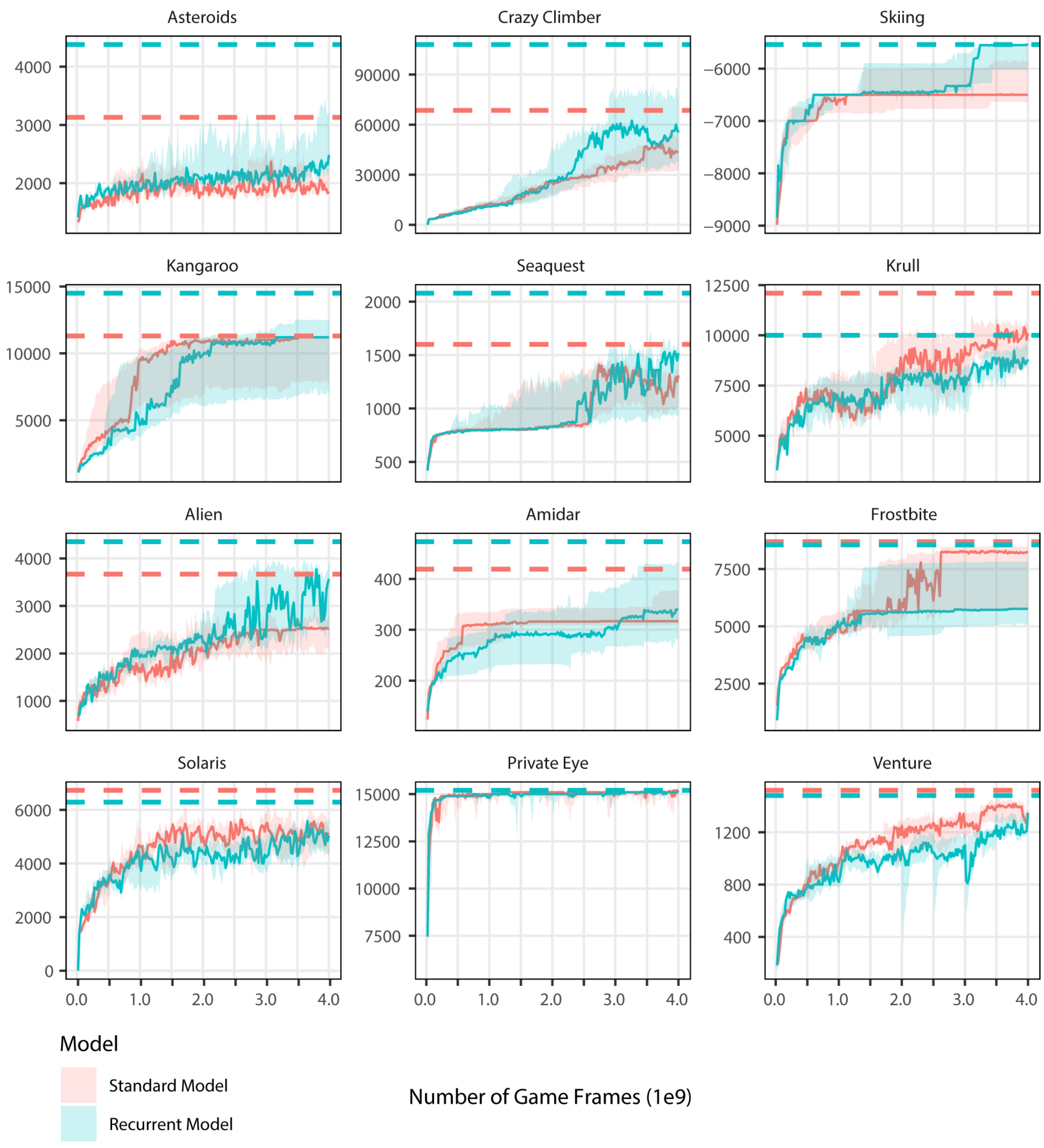


Figure 7.2: **Original and Recurrent model's performance across generations on Atari 2600 games.** Highest scores achieved by both are plotted as dashed lines. The median is plot with a solid line and 95% bootstrapped confidence intervals of the median across 5 experiments of the current elite per run, where the score for each elite is a mean of 30 no-op independent episodes.

shown, human-optimal, score exploit and dense reward categories each saw some positive benefit to the overall results using a recurrent convolutional layer. However, not all games experienced a positive end result. The category of Sparse Rewards saw either no change or a slight degradation to performance with the use of a recurrent convolutional layer. Skiing saw no additional benefit. Frostbite saw a small degradation in performance. Krull saw a comparably large percentage difference.

Figure 7.2 provides complementary information to table 7.2, it provides the elite's score through each stage of the 4 billion frames run. Also providing dashed lines for the highest achieved score from both approaches. These results are in interesting contrast to those presented in table 7.2. The evolutionary process uses a mutation rate not pre-optimised for the task and as a result certain games see a high level of volatility in score performance; (see crazy climber and alien). To ensure the results are not inaccurate to the overall performance the highest score is also recorded and not just the end score where agents could be in a stage of degradation. It's important to provide context for each result, the difference in scores is only notable if a deviation from one score to another is a results of multiple meaningful changes, as opposed to a minimal change but with huge performance gains. Therefore each result with a notable percentage difference (above 10%) and a statistically significant ($p < 0.05$) difference, is examined. Everything discussed here can be followed along with video examples from the Atari Zoo platform[2]. A description of the game and the scoring system is provided.

Asteroids is a human-optimal environment which saw the greatest positive change

[2]https://eng.uber.com/atari-zoo-deep-reinforcement-learning/

from the inclusion of a recurrent convolutional layer; with a 54% increase in elite score. The aim of asteroids is to avoid obstacles while also firing projectiles to destroy them. To obtain a greater score agents must destroy the obstacles. The score associated with each obstacle depends on their size. Scores range from 20 to 100 at these early stages. So, with a score deviation of 1570 agents, at a minimum, would require destroying 16 additional obstacles over the standard architecture. In motion, it is difficult to pinpoint the exact behavioural change. Both adopt a strategy of remaining stationary while turning in either a clockwise or anti-clockwise direction while firing. This intern provides more opportunity to strike obstacles, although neither show any sign of deliberate motion to avoid obstacles.

Crazy Climber is also a human-optimal task which produced a higher final score with the use of a recurrent layer, but figure 7.2 shows how the recurrent model produced consistently greater scores in this environment. The environment tasks an agent to climb to the top of four skyscrapers while avoiding multiple obstacles; the challenge comes from avoiding objects being dropped and obstacles which require timing, at least at early stages. Each floor the agent passes receives 100 points and each skyscraper is 160 floors high. A bonus is received if the agent reaches the top of the skyscraper and enters the helicopter (10^4 multiplied by skyscrapers level); which requires the agent to be in the correct position. The bonus reduces in value every 10 seconds that the agent spends traversing the building. A score difference of 16000 suggests the recurrent model is achieving one skyscraper further than the standard model on average. Though it can not exactly be quantified, due to the bonus being dependent on the time spent per skyscraper, the recurrent model is avoiding a greater number of obstacles and finding greater appropriate paths to progress upwards. In figure 7.2, it shows that with crazy climber the recurrent network sees a surge in score performance deviation at 3 billion frames. With this knowledge, the suggestion is that at this stage a number of runs are achieving performance to the third skyscraper. But, the standard architecture's highest score suggests at some point it also achieved the third skyscraper but did not keep that behaviour consistent.

Kangaroo is a score exploit environment and saw a 24% greater end score with the recurrent model. Kangaroo's evolution process sees the medium producing comparable results to that of the standard model, but with a wider deviation into greater and worse performance. In the environment, there are four floors in which the agent must traverse upwards to reach the top floor while avoiding the enemies throwing projectiles. Sometimes projectiles are thrown and must be jumped over and sometimes they are thrown so that the agent must duck. Enemies traverse to the agent and when killed provide a score. This aspect is the reason this is considered a scoring exploiting environment. A majority of reinforcement learning and this approach exhibit the behaviour of punching enemies as they get close. Punching enemies grants a 200 score increase and also a 200 score increase for avoiding projectiles. As behaviours are limited to 200 score increments per desirable action, the score difference of 3200 indicates 16 additional desirable moves were taken by the recurrent convolutional network. An interesting behaviour emerged in 2 recurrent runs, in which after punching an enemy they would jump backwards and then duck. Though it did not seem to affect the score (all the scores were achieved by punching the enemies) it could have been a beneficial trait to avoid the projectiles dropped. This is notable as it is not seen in any of the standard architecture runs.

Seaquest is an environment set underwater in which an agent has to save divers and shoot enemies while maintaining the oxygen supply. When oxygen runs low, agents should manoeuvre to the top of the environment to replenish oxygen, but if they have not collected a diver then agents lose a life. If oxygen runs out agents lose a life. Agents have three lifes. Each diver collected is worth 100 points and each enemy killed is worth 20 points. The difference between the two scores implies that 13 more enemies were killed over the original architecture or one additional diver was rescued with 3 enemies killed. Previously, Seaquest has seen to work poorly with GAs due to a sub optimal behaviour. This behaviour keeps agents at the bottom of the ocean until they run out of oxygen [227]. However, both the standard and recurrent architecture did exhibit the behaviour of surfacing for oxygen in our runs. This can be seen in the jump at 2.5 billion frames from a score of sub 1000 to instantly jumping above it.

Krull is the only environment in which the standard model has a noticeable improvement over the recurrent model. This can be seen in table 7.2 but also in figure 7.2, the recurrent model is consistently below the standard model after 2 billion frames. Krull takes place on four separate screens, each with opportunities to receive points. However, agents usually stay on the second screen called the *'lair of the Widow of the Web'* in which agents avoid an enemy while traversing through a webbed environment. Every movement in the webs give agents a varying score but can be as much as 99 points per movement. Many strategies, including GAs, exploit this sequence by continually moving within the webs instead of progressing in the game. As a result, it is difficult to deduct the differences in behaviours that make the standard model superior in this task. A common strategy was to leave the screen via either the left or right and this would restart the section; if the agent was in the environment too long then the enemy would pursue them aggressively. Leaving the scene and returning prevents this and allows the score exploit to continue.

Alien is the only hard exploration task that saw a sizeable delta in the performance of the two models. Alien is a maze type environment in which the agent must collect all the dots in a stage while avoiding three enemies to continually progress. Each dot scores the agent 10 points. There is a bonus item in the same position each game worth 50 points. A power dot can be picked up which allows a temporary ability to kill enemies; this provides 1000 points if enemies are killed. The majority of strategies found by both the standard and recurrent is to collect the power dot and kill enemies to maximise points; neither deliberately pick up dots. The score difference of 1020 suggests that the recurrent model picks up 2 extra dots and kills an extra enemy. In motion, both learn the significance of traversing to the top of the maze to access the power cell. This usefully leads to two enemies being killed at that stage as agents stay in their corners as the enemies come to them and receive the points bonus. Where the recurrent model seems to adjust is, with certain runs, the agents proceed to the bottom left corner where the power cell re-emerges and this is where further enemies are killed.

7.4 Discussion

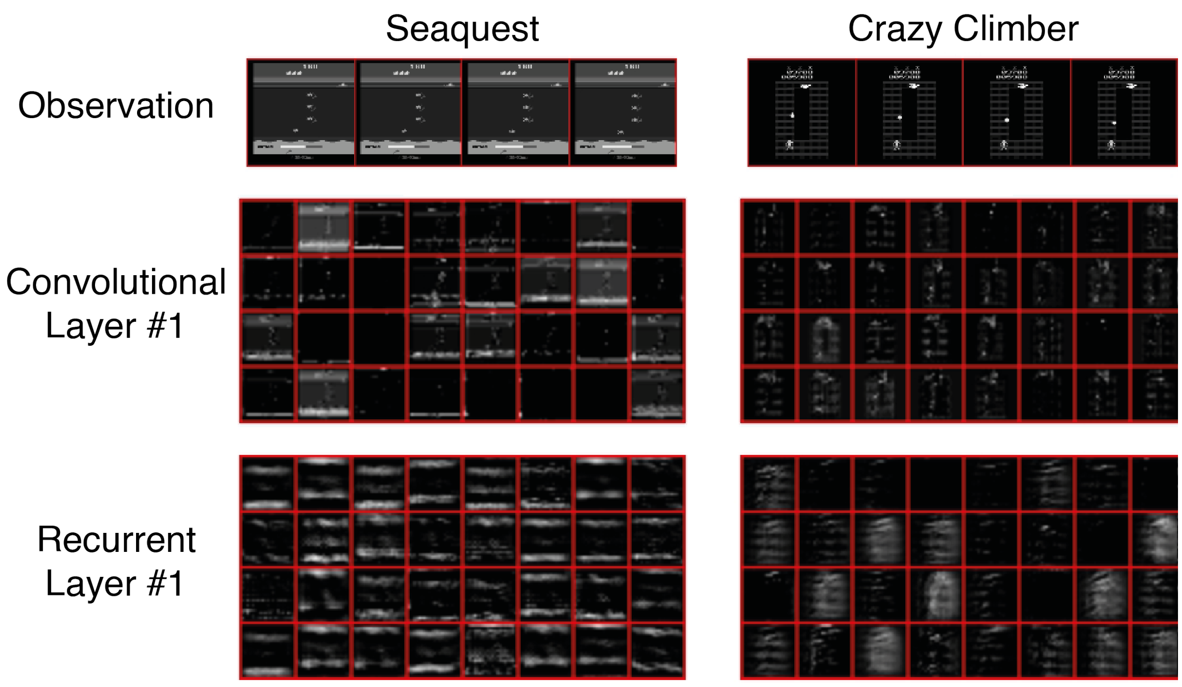


Figure 7.3: **Activation visualisation of the recurrent model on Seaquest and Crazy Climber.** The figure shows a still video frame from trained recurrent model agents. The observation layer shows the processed inputs from the environment. The next layer shows the activation features for the convolutional layer. Then, the activation features for the recurrent convolutional layer.

In terms of score, the results section shows the recurrent model demonstrated: 5 beneficial score effects (over 10% improvement); 5 indifferent score effects (0-9% change) and 1 detrimental score effect (over 10% decrease). The recurrent model produced no notable change in sparse reward tasks and only 1 within dense reward. Therefore, it appears the recurrent model does not aid in traversal in hard exploration tasks. However, 5 of the 6 easy exploration saw a significant change. To further investigate, this section utilises the Atari Zoo platform to visualise real-time features within the recurrent layer; as well as inspecting the behaviours of agents.

Figure 7.3 demonstrates a recurrent layer on the Seaquest and Crazy Climber game. In relation to traditional image processing, features in the recurrent layer (figure 7.3) resemble typical blurring/smoothing where image detail and noise are reduced via a low-pass filter. There are different types of blurs, all of which standard convolution can achieve, with the correct filter. Though it is not exact, the blurring may bias to Gaussian-like blurs; where each pixel is a result of a weighted average of its neighbours, and farther pixels have decreasing weight. The recurrent implementation (see equation 5.1) uses an identity kernel passthrough from iteration 0, so the centre activation biases towards the largest activation during convolution, like a Gaussian blur. So, objects will not lose their spatial location in the feature space; this can be seen in Seaquest with the river bed and the building in Crazy Climber. The spread of a blur is dependent on the kernel size; however, blurring an image multiple times with a small width Gaussian-kernel is equivalent to blurring once with a larger width Gaussian-kernel [268]. This could be a potential benefit recurrency offers, as a larger kernel requires a greater number of weights to correctly fit the kernel blur pattern. Visually, in Crazy Climber, the gaps for the windows are still present, seen via the horizontal changes in patterns, but the building itself becomes a greater activation area. During playback, when objects are dropped by enemies, a greater number of cells are active surrounding the object. This could provide feedback to influence behaviours, as slight blurs have shown to lead to noticeably different activations in later layers compared to an un-blurred input [52]. Also, it is not uncommon to pre-process images with blurs for edge detection purposes [135, 184].

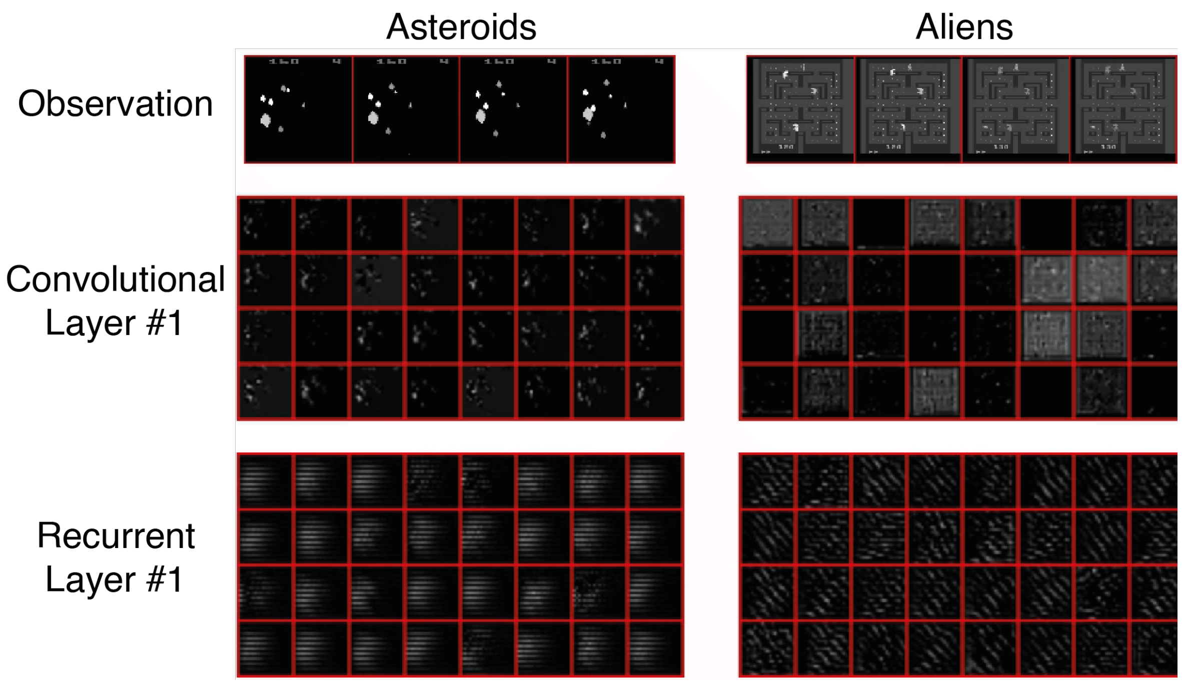


Figure 7.4: **Activation visualisation of the recurrent model on Asteroids and Aliens.** The figure shows a still video frame from trained recurrent model agents. The observation layer shows the processed inputs from the environment. The next layer shows the activation features for the convolutional layer. Then, the activation features for the recurrent convolutional layer. Superior results were achieved over the standard model.

Figure 7.4 shows the recurrent layer within Asteroids and Alien, two environments which produce clear benefits with recurrency. The visualisation of the recurrency layer seems to suggest Gabor-like feature outputs. Gabor filters are linear filters that have long been associated with image processing for feature extraction, texture segmentation and texture analysis. A convolutional Gabor filter is a product of a Gaussian kernel function modulated by a sinusoidal plane wave. By adjusting Gabor filters at specific frequencies and directions, these filters can offer the desirable properties of spa-

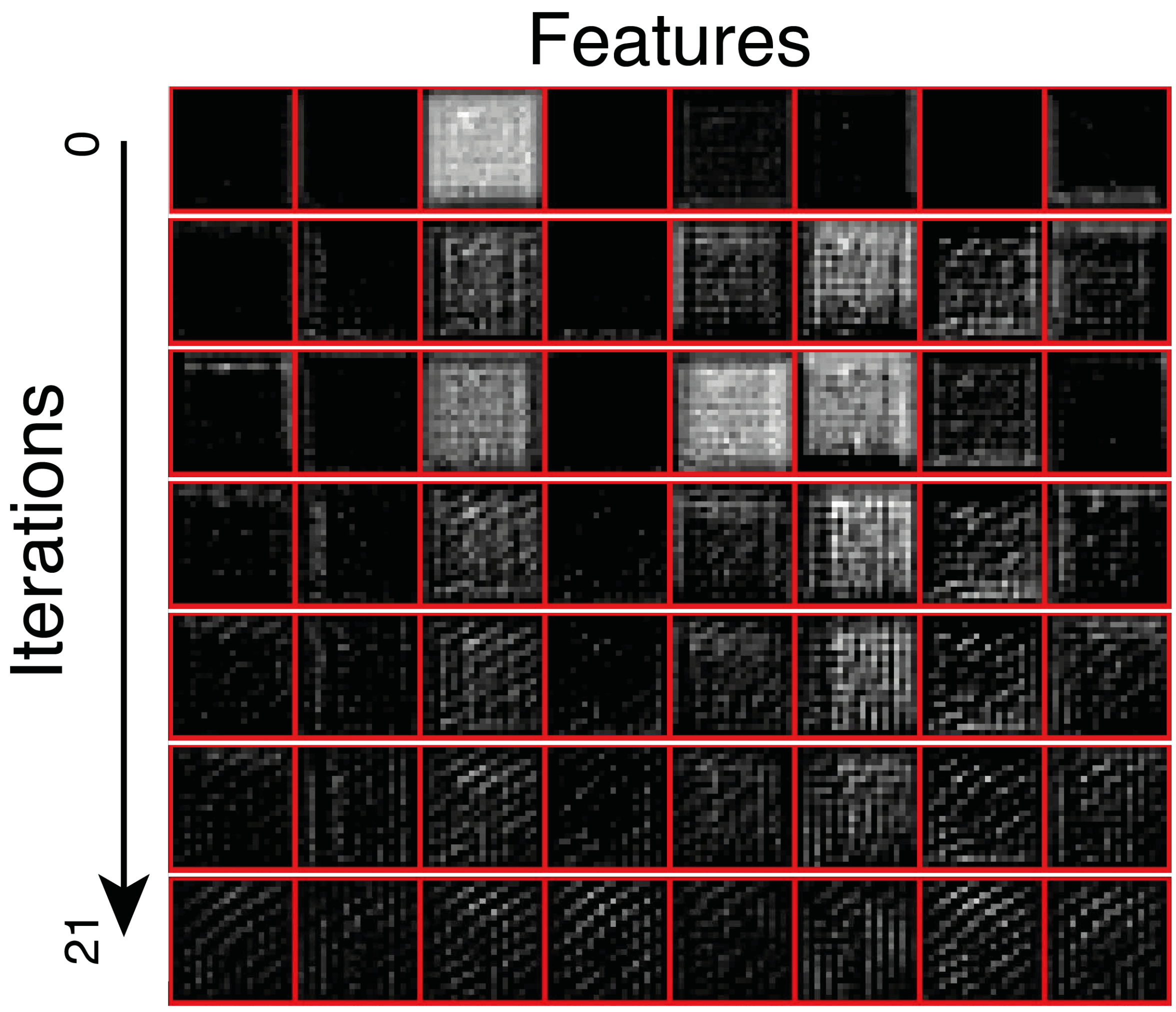


Figure 7.5: **Activation visualisation for the recurrent convolutional process**. The features are produced on Alien with an agent trained to 4 billion frames. The top layer are features taken from the convolutional layer. Each layer down is the visualisation of the same features after a number of iterations on the recurrent convolutional layer. This shows how features can be 'flooded' with repeated convolution.

tial locality and orientational selectivity [80]; and have been used across various image processing domains, such as face recognition [225, 11] and text extraction [182]. Gabor filters have been used in convolutional neural networks previously as fixed weight kernels to extract intrinsic features [195], or as a pre-processing stage [23, 122]. Even when the kernels weights are controlled via supervised learning, Krizhevsky et al. showed that deep ConvNet's, trained on real-life images, tend to populate mostly Gabor-like filters in the first convolutional layer [121]. So, it would not be uncommon to see these results.

However, in Such et al.'s Atari zoo work, it shows that kernels for evolutionary algorithms appear less regular (and even random) than their gradient-based counterparts, which often have spatial structure and sometimes resemble edge detectors [227]. Further, figure 7.5 shows the impact of the iterative process of recurrent convolution on the features, from iteration 0 to 21. A feature set that starts unique becomes fairly homogeneous towards the end. It is for these reasons the production of Gabor-like features are unlikely and instead the feature space is becoming saturated. In Meng and Yang's work with the shunting equation, the inspiration for recurrent convolution, the discussion of parameter sensitivity for the *'passive decay rate'* shows that if too low, the neuron activity saturates quickly and the activity space is flooded. For the recurrent convolutional process, the *'iteration parameter'* would produce the same impact; enough iterations and the feature space becomes saturated. Comparing figure 7.3 and 7.4, it is clear that this does not affect every game environment and the iteration parameter would need to be tuned game to game. Therefore, incorporating this parameter into the evolutionary process may allow each model to choose the parameters which suits it best; which may involve turning off the recurrent layer due to the statistical negative result on some games. Despite not fully understanding the purpose recurrency offers for Asteroids and Alien, the results offer a significant improvement.

Table 7.3 shows the max scores taken from multiple domains and plotted against comparable strategies, that also achieved scores from 30 independent no-op episodes.

Crazy Climber

		Scores
GA Standard		**68600**
Gorila DQN	[170]	85919.16
DDQN	[243]	101874
GA Recurrent		**108000**

Kangroo

		Scores
GA Standard		**11300**
C51	[14]	12853
Distributional DQN	[98]	12853
Reactor ND	[84]	13349
DDQN	[243]	13651
GA Recurrent		**14500**

Seaquest

		Scores
GA Standard		**1600**
IMPALA (shallow)	[59]	1716.9
A3C	[63]	1744
IMPALA (deep)	[59]	1753.2
A2C	[260]	1754
ACKTR	[260]	1776
GA Recurrent		**2080**

Asteroids

		Scores
GA Standard		**3130**
NoisyNet DQN	[63]	3455
IMPALA (shallow)	[59]	3508.1
Reactor	[84]	3726.1
QR-DQN-1	[40]	4226
GA Recurrent		**4380**

Alien

		Scores
GA Standard		**3670**
DDQN	[14]	3747.7
PDD DQN	[249]	3941
Distributional DQN	[98]	4055.8
Reactor ND	[84]	4199.4
GA Recurrent		**4350**

Table 7.3: **Comparing performance between algorithms and architectures on multiple ALE games.** Only the games that the recurrent model produced a noticeable performance benefit are shown. Bold indicates the GA standard and recurrent models. Scores are arranged in ascending order. Each learning style has a citation to the original source.

No-op starts have a non-deterministic starting position where the agent selects the "do nothing" action for up to 30 times at the start of an episode. The recurrent model would be an improvement on a number of established architectures and learning techniques.

7.5 Conclusion

This chapter implemented a novel recurrent convolutional layer to a conventional ConvNet and was able to produce performance improvements on a number of domains in ALE using deep neuroevolution. 12 Atari game environments were trained for 4 billion frames on 5 evolutionary runs. 6 produced a statistically improved score, 4 produced a statistically negative impact on score and the remaining 2 produced no change. However, the largest difference in negative score was 13% vs the largest improvement of 54% increase in performance. Benefits were seen across all easy exploration tasks and on hard exploration dense reward tasks. No benefits were seen on the selection of sparse reward tasks, and 2 of the 3 saw a negative impact on score. Of the games that were visually analysed, a majority produced a noticeable difference in agent's behaviour that was seen across many evolutionary runs when compared to the conventional ConvNet architecture. Due to computational constraints, only 12, of the possible 54, games were analysed and the maximum frames were limited to 4 billion instead of the original 6 billion. So, there is still an avenue to investigate on the ALE platform; however, this chapter has implications outside of ALE. Firstly, recurrent convolution is not solely restricted to producing motion planning landscapes. It can now be explored in the many other environments which benefit from large ConvNets. Secondly, the recurrent convolution process is not tied to neuroevolution and is an architectural change; this allows any other learning style to adapt to this same architecture. This may be particularly interesting as, with the recurrent layer, other established learning styles and architectures were outperformed in 5 environments. Finally, the recurrent convolutional layer allows flexibility in its implementation. An aspect which was discovered during the analysis of the ConvNet's features show that recurrency, in certain game environments, causes uniformity across the features. It was speculated that the iteration parameter was the main factor and investigating further could offer an insight into how this parameter should be tuned. In a similar vein, features like dropout could be incorporated to prevent the convergence of features, allowing the previous layer to pass through without recurrency applied. This is to show that there is still room to further experiment with this architecture and those above provide a few ideas.

8 | Conclusions

8.1 Summary of Conclusions

The goal of the thesis was to examine the evolution of motion planning systems that exhibit robust, deliberate, and efficient behaviour for use in artificial evolutionary systems. Throughout the thesis, there is a focus on moving away from domain-specific architectures, as well as investigating these solutions on sparse reward tasks in an attempt to eliminate innate bias towards explicit desirable solutions. This study presents its findings from an artificial life simulation model and a standard multi-environment platform.

Chapter 4 applies our sophisticated neuroevolution understandings to a strict motion planning task within an artificial life simulation model. Both chapters investigate the scalability and quality of motion planning solutions via direct and indirect encoding. The belief was that direct encoding would not be suitable due to the high dimensionality of the domain problem. Whereas, indirect encoding could produce the repeatable and scalable patterns necessary for motion planning. Agents must converge on multiple locations in an efficient manner to be successful. The results confirm that direct encoding would not be suitable, with even the smallest domain receiving inconsistent outcomes. Indirect encoding did provide adequate solutions at the original domain size but declined in the quality of movement when compared to static solutions. Thus, chapter 4 indicates that within neuroevolution there is a lack of distinguished approaches for motion planning that compete with static artificial potential field methods.

Chapter 5 uses the results of the previous chapter to construct a novel use of convolutional neural networks to overcome the issues in neuroevolution planning. This chapter uses the same domain and task as in chapters 4. With a simple genetic algorithm and the novel architecture, results show greater behaviour in all categories (robustness, deliberateness, and efficiency) over previous results. In comparison to the static model, there is a slight reduction in robustness but greater efficiency. A key component of the model's success relies on the fixed genome size. This restricts the domain to a low dimensional space. While still spreading activations from local areas to span the greater landscape.

Chapter 6 extends the work presented in chapter 5 by assessing if motion planning can evolve with various network combinations. Previous chapters restrict the task domain with static networks. With greater evolutionary freedom the task domain presents greater difficulty. Further, the network designs are adjusted to remove inherent domain-specific influence. The chapters focus was on how/if evolved motion planning could operate successfully in domains with greater evolutionary difficulty. The outcome confirms that evolved motion planning, with a simple GA, can adapt to many different evolutionary scenarios of different complexity including the most ambitious combination which strips the majority of domain influence without fundamentally changing the architecture. A state-of-the-art neuroevolution general game player is presented to the task domain as a comparison. These results show that it is difficult to achieve the same behaviour presented without the use of explicit motion planning.

Finally, Chapter 7 addresses whether the recurrent convolutional neural network is restricted to the task domain this thesis has been positioned around. To investigate this, a prominent multi-environment platform is used for evaluation. The platform has commonly been utilised with convolutional neural network architectures. The recurrent convolutional neural network is simulated in twelve diverse environments. Results show that recurrent convolution can expand to other environments. Five of the environments show an improved end score with the recurrent model. Behaviours ranged from unnoticeable to observable strategy differences. The best results, per domain, are compared to alternative learning strategies and architectures. The recurrent convolutional neural network has an improved score on several different approaches.

8.2 Contribution of this work

- Provides a new neuroevolution approach for adopting motion planning in artificial evolutionary systems (Chapters 4). The model allows unbounded growth of the configuration space. This is counter to those with static implementations or tight evolutionary bounds. The implementation uses an indirect encoding configuration. By achieving deliberate motion in an artificial life domain, it proved unconstrained motion planning is possible with current neuroevolution techniques.

- Implemented a novel extension to a convolutional neural network with the use of recurrent convolution to target motion planning (Chapter 5). In the domain, the model saw greater robustness than any previous neuroevolutionary approach; and greater efficiency than any static or neuroevolutionary precedent. This demonstrates that the unique architecture provides a notable improvement in motion planning. This was achieved with a standard conventional genetic algorithm.

- Establishes the advantage of a motion planning component in a general-purpose network when compared to a state-of-the-art deep convolutional network (Chapters 6). The domain presented requires long term deliberative planning to succeed, while providing sparse rewards. The deep convolutional network could not achieve the long term behaviours. When the convolution planning network is utilised with other networks the deliberative behaviour becomes obtainable. This was even shown to be true when removing a majority of its domain-specific network elements. Thus demonstrating the capability of the architecture as well as assuring motion planning versatility with greater task difficulty.

- Demonstrates that recurrent convolution within a deep convolutional network can outperform other learning techniques and architectures on multiple domains (Chapter 7). This illustrates how recurrent convolution can aid in scenarios which are not restricted to, but not excluding, motion planning in a variety of task domains. This was achieved on a standard benchmarking platform.

8.3 Limitations & Future Work

Throughout the thesis, different learning styles and architectures are compared directly. The author sought to create a fair comparison when this arises. However, experimentation is the judgement of this author and could cause unintentional oversight. When comparing HyperNEAT, recurrent ConvNets, and Large ConvNets there are greater factors to consider. Each has many parameters and different dimensionalities. Therefore, each could require vastly different considerations to achieve their optimal performance. To overcome this issue, each section seeks to provide a variety of parameters and configurations. This aims to cover a diverse set of particular behaviours. The reasoning for each choice is explained in each chapter. But, it can not be said that one approach could not achieve a specific behaviour unequivocally. For example, chapter 2 provides many ways to improve the evolutionary search. Yet, the majority of this work utilises a simple GA. By improving evolutionary search, results could improve across all strategies. This, however, is not the scope of this thesis and can be seen as future work.

Future work can also be done for motion planning in alternative artificial life domains. Those models which utilise artificial potential field type motion planning can simply implement the recurrent ConvNet from this work. Additionally, the domain used for the majority of this thesis has proven to be a difficult task for general systems. Further investigation could be made about what methods could be taken from the general hybrid model and be lifted to greater established networks, as this work did in chapter 7. Finally, the positive results of recurrent convolution on multiple ALE games allows opportunities to evaluate alternative learning techniques such as RL. As well as applying this process to domains which have already drawn success from conventional ConvNets.

Acronyms

A3C Asynchronous Advantage Actor-Critic. 75

AGE Analog Genetic Encoding. iii, 33–35

ALE Arcade Learning Environment. iv, viii, 46, 68, 75, 76, 143–146, 154, 155, 170, 171, 178

ANN Artificial Neural Networks. vii, 22, 26, 37, 40, 41, 48, 64, 83

BF-NEAT Bloat-free NEAT. 23

CE Cellular Encoding. iii, 32, 33

CGP Cartesian Genetic programming. iii, vii, 35–37

CGPANN Cartesian Genetic Programming of Artificial Neural Networks. 37

CNE Conventional NeuroEvolution. 9, 50

ConvNet Convolutional Neural Network. viii, 41, 46–49, 76, 106–110, 112, 118, 120, 125, 130, 134, 141–143, 145, 148, 150–157, 169, 171, 172

CoSyNE Cooperative Synapse NeuroEvolution. iii, vii, 18, 27–29, 65

CPPN Compositional Pattern Producing Networks. iii, vii, 38–45, 65, 84, 94, 95, 99, 102

DDQN Double DQN. 75

DN Decision Network. iv, 69, 70, 72, 73, 79, 80, 105, 106, 108–110, 113, 114, 118, 120–122, 124, 126

DQN Deep Q Networks. viii, 75, 76, 144, 145

EAs Evolutionary Algorithms. 9, 10

ES-HyperNEAT Evolvable-substrate HyperNEAT. 44, 45

ESP Enforced SubPopulations. iii, 16–18, 27, 65, 66

GA Genetic Algorithm. v, 9, 10, 21, 31, 50, 52, 76, 107, 111, 118, 153, 154, 163, 164

GAR Galactic Arms Race. 40

GNARL GeNeralized Acquisition of Recurrent Links. iii, vii, 13, 14, 19, 23

HSANE Hierarchical SANE. 16, 18

HybrID Hybridized Indirect and Direct encoding. 42, 43, 88

HyperNEAT Hypercube-based NEAT. iii, vii, viii, 40–45, 60, 65, 66, 75, 77–82, 84, 86, 88, 90, 93–95, 98, 99, 101–106, 114–117

L-system Lindenmayer systems. iii, vii, 30–32

LSTM Long short-term Memory. 18

MOEAs Multi-objective evolutionary algorithms. iv, 56, 60

NE Neuroevolution. 9, 10, 13, 16, 18, 26, 33, 46, 50, 60, 64, 66, 76, 143, 144

NEAT NeuroEvolution of Augmenting Topologies. iii, vii, viii, 16, 18, 20–23, 25, 27, 35, 38, 43, 44, 60, 65, 66, 77–86, 90, 91, 93, 94, 104

NERO Neuro Evolving Robotic Operatives. 25

NevA NeuroEvolutionary Algorithm. iii, 26, 27

NS Novelty Search. iv, 59–61

NSGA Non Dominated Sorting Genetic Algorithm. 57

NSGA-II Non Dominated Sorting Genetic Algorithm-II. 57

QD Quality Diversity. iv, 60–62

RC Task River Crossing Task. iv, v, viii, ix, 67–74, 77–80, 82, 89, 94–96, 102, 108, 111, 113, 118, 120, 123, 125, 126, 131, 140, 143–147, 149–151

RCGPANN Recurrent Cartesian Genetic Programming of Artificial Neural Networks. 37

ReLU Rectified Linear Unit. 49, 81, 108, 110, 111, 117, 125, 141

ResNets Residual Networks. 49

RL Reinforcement learning. 3, 46, 143, 144, 153, 154, 178

rtNEAT Real-Time NEAT. 25, 26

SAGA Species Adaptation Genetic Algorithms. 54

SANE Symbiotic Adaptive NeuroEvolution. iii, 14–18, 27, 65

sGA Structured Genetic Algorithm. iii, vii, 11–13

SGD Stochastic Gradient Descent. 8, 46

SM Shunting Model. iv, 68, 72, 73, 77, 78, 80, 82, 95, 96, 100, 101, 103–108, 110, 113, 117, 118, 121

TWEANN Weight Evolving Artificial Neural Network. 11, 16, 18, 20, 65, 83

VEGA Shaffer's Vector Evaluated Genetic Algorithm. 56, 57

Bibliography

[1] T. Aaltonen, J. Adelman, T. Akimoto, M. Albrow, B. A. González, S. Amerio, D. Amidei, A. Anastassov, A. Annovi, J. Antos, et al. Measurement of the top-quark mass with dilepton events selected using neuroevolution at cdf. *Physical review letters*, 102:152001, 2009.

[2] V. Abazov, B. Abbott, M. Abolins, B. Acharya, M. Adams, T. Adams, E. Aguilo, M. Ahsan, G. Alexeev, G. Alkhazov, et al. Observation of single top-quark production. *Physical Review Letters*, 103:92001, 2009.

[3] A. A. Agrachev and Y. Sachkov. *Control theory from the geometric viewpoint*. Springer Science & Business Media, 2013.

[4] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5: 54–65, 1994.

[5] M. S. Arun Kumar Sangaiah and Z. Zhang. *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*. Elsevier, 2018.

[6] Y. M. Assael, B. Shillingford, S. Whiteson, and N. de Freitas. LipNet: Sentence-level Lipreading. *CoRR*, abs/1611.01599, 2016.

[7] J. E. Auerbach. Automated evolution of interesting images. Technical report, MIT Press, 2012.

[8] J. E. Auerbach and J. C. Bongard. Evolving complete robots with cppn-neat: the utility of recurrent connections. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1475–1482. ACM, 2011.

[9] P. Baldi and P. Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 83:51–74, 2016.

[10] A. Baldominos, Y. Saez, and P. Isasi. Hybridizing evolutionary computation and deep neural networks: An approach to handwriting recognition using committees and transfer learning. *Complexity*, 2019, 2019.

[11] T. Barbu. Gabor filter-based face recognition technique. *Proceedings of the Romanian Academy*, 11(3):277–283, 2010.

[12] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[13] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[14] M. G. Bellemare, W. Dabney, and R. Munos. A Distributional Perspective on Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

[15] S. Benson-Amram and K. E. Holekamp. Innovative problem solving by wild spotted hyenas. *Proceedings of the Royal Society B: Biological Sciences*, 279: 4087–4095, 2012.

[16] E. J. W. Boers, H. Kuiper, B. L. M. Happel, and I. G. Sprinkhuizen-Kuyper. Biological Metaphors In Designing Modular Artificial Neural Networks. In S. Gielen

and B. Kappen, editors, *ICANN '93*, pages 780–780. Springer London, 1993. ISBN 978-1-4471-2063-6.

[17] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars. *CoRR*, abs/1604.07316, 2016.

[18] J. Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108:1234–1239, 2011.

[19] J. C. Bongard and R. Pfeifer. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 829–836. Morgan Kaufmann Publishers Inc., 2001.

[20] J. Borg, A. Channon, and C. Day. Discovering and Maintaining Behaviours Inaccessible to Incremental Genetic Evolution Through Transcription Errors and Cultural Transmission. In *Advances in Artificial Life, ECAL 2011: Proceedings of the Eleventh European Conference on the Synthesis and Simulation of Living Systems*, pages 101–108. MIT Press, 2013.

[21] J. M. Borg and A. Channon. Evolutionary adaptation to social information use without learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10199 LNCS, pages 837–852. Springer, 2017.

[22] S. P. Brooks and B. J. T. Morgan. Optimization Using Simulated Annealing. *The Statistician*, 44:241, 2006.

[23] G. S. Budhi, R. Adipranata, and F. J. Hartono. The use of gabor filter and back-propagation neural network for the automobile types recognition. In *2nd International Conference SIIT 2010*, 2010.

[24] Y. Burda, H. Edwards, A. J. Storkey, and O. Klimov. Exploration by Random Network Distillation. *CoRR*, abs/1810.12894, 2018.

[25] N. Burtnyk and M. Wein. Computer-generated key-frame animation. *Journal of the SMPTE*, 80:149–153, 1971.

[26] L. Cardamone, D. Loiacono, and P. L. Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, page 1179. ACM, 2009.

[27] R. Chandra, M. Frean, and M. Zhang. An encoding scheme for cooperative coevolutionary feedforward neural networks. In *Australasian Joint Conference on Artificial Intelligence*, pages 253–262. Springer, 2010.

[28] R. Chandra, M. Frean, M. Zhang, and C. W. Omlin. Encoding subcomponents in cooperative co-evolutionary recurrent neural networks. *Neurocomputing*, 74: 3223–3234, 2011.

[29] T.-Y. Chang, S.-W. Kuo, and J.-J. Hsu. A two-phase navigation system for mobile robots in dynamic environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 1, pages 306–313. IEEE, 1994.

[30] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of robot motion: theory, algorithms, and implementation.* MIT press, 2005.

[31] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pages 2764–2771. IEEE, 2009.

[32] J. Clune, C. Ofria, and R. T. Pennock. The sensitivity of HyperNEAT to different geometric representations of a problem. In *the 11th Annual conference*, page 675. ACM Press, 2009.

[33] J. Clune, B. E. Beckmann, R. T. Pennock, and C. Ofria. HybrID: A hybridization of indirect and direct encodings for evolutionary computation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5778 LNAI, pages 134–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 9 2011.

[34] J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15:346–367, 2011.

[35] P. Coates, T. Broughton, and H. Jackson. Exploring three-dimensional design worlds using lindenmayer systems and genetic programming. *Evolutionary design by computers*, pages 323–341, 1999.

[36] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. Stanley, and J. Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems*, pages 5027–5038, 2018.

[37] V. Coverstone-Carroll, J. Hartmann, and W. Mason. Optimal multi-objective low-thrust spacecraft trajectories. *Computer methods in applied mechanics and engineering*, 186:387–402, 2000.

[38] G. Cuccu and F. Gomez. When novelty is not enough. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6624 LNCS, pages 234–243. Springer Berlin Heidelberg, Berlin, Heidelberg, Apr. 2011.

[39] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521:503–529, 2015.

[40] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. *CoRR*, abs/1710.10044, 2017.

[41] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit Quantile Networks for Distributional Reinforcement Learning. *CoRR*, abs/1806.06923, 2018.

[42] D. D'Ambrosio, J. Lehman, S. Risi, and K. O. Stanley. Evolving Policy Geometry for Scalable Multiagent Learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 731–738. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[43] D. B. D'Ambrosio and K. O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 974–981. ACM, 2007.

[44] D. B. D'Ambrosio and K. O. Stanley. Scalable multiagent learning through indirect encoding of policy geometry. *Evolutionary Intelligence*, 6:1–26, 2013.

[45] D. Dasgupta and D. R. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96. IEEE, 1992.

[46] D. Dasgupta and D. R. McGregor. Nonstationary Function Optimization using the Structured Genetic Algorithm. In *PPSN*, pages 145–154. Citeseer, 1992.

[47] D. Dasgupta and D. R. McGregor. *sGA: a structured genetic algorithm.* Citeseer, 1993.

[48] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[49] K. A. De Jong. *Evolutionary computation: a unified approach*. MIT press, 2006.

[50] K. Deb. Multi-objective evolutionary algorithms. In *Springer Handbook of Computational Intelligence*, pages 995–1015. Springer, 2015.

[51] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6:182–197, 2002.

[52] S. Dodge and L. Karam. Understanding how image quality affects deep neural networks. In *2016 eighth international conference on quality of multimedia experience (QoMEX)*, pages 1–6. IEEE, 2016.

[53] G. Dozier, S. McCullough, A. Homaifar, E. Tunstel, and L. Moore. Multiobjective evolutionary path planning via fuzzy tournament selection. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 684–689. IEEE, 1998.

[54] J. Drchal, J. Koutnik, and M. Snorek. HyperNEAT controlled robots learn how to drive on roads in simulated environment. In *2009 IEEE Congress on Evolutionary Computation*, pages 1087–1092, 2009.

[55] P. Dürr, C. Mattiussi, and D. Floreano. Neuroevolution with Analog Genetic Encoding. In *Parallel Problem Solving from Nature - PPSN IX*, pages 671–680. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[56] R. Dybowski, V. Gant, P. Weller, and R. Chang. Prediction of outcome in critically ill patients using artificial neural network synthesised by genetic algorithm. *The Lancet*, 347:1146–1150, 1996.

[57] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-Explore: a New Approach for Hard-Exploration Problems. *CoRR*, abs/1901.10995, 2019.

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

[59] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018.

[60] J. Euchner. Innovation Engines. In *Research-Technology Management*, volume 58 of *Automated Creativity and Improved Stochastic Optimization via Deep Learning*, pages 9–10. ACM, 2015.

[61] J. Fekiač, I. Zelinka, and J. C. Burguillo. A review of methods for encoding neural network topologies in evolutionary computation. In *Proceedings of 25th European Conference on Modeling and Simulation ECMS*, pages 410–416, 2011.

[62] D. B. Fogel and L. C. Stayton. On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems*, 32:171–182, 1994.

[63] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy Networks for Exploration. *CoRR*, abs/1706.10295, 2017.

[64] I. Gabriel, V. Negru, and D. Zaharie. Neuroevolution based multi-agent system for micromanagement in real-time strategy games. In *Proceedings of the Fifth Balkan Conference in Informatics*, page 32. ACM, ACM, 2012.

[65] J. Gauci and K. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 997–1004. ACM, 2007.

[66] J. Gauci and K. O. Stanley. A Case Study on the Critical Role of Geometric Regularity in Machine Learning. *Artificial Intelligence*, pages 628–633, 2008.

[67] J. Gauci and K. O. Stanley. Indirect encoding of neural networks for scalable go. In *International Conference on Parallel Problem Solving from Nature*, pages 354–363. Springer, 2010.

[68] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323. PMLR, 2011.

[69] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3:95–99, 1988.

[70] D. E. Goldberg, J. Richardson, et al. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[71] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[72] F. Gomez and R. Miikkulainen. 2D pole balancing with recurrent evolutionary networks. In *International Conference on Artificial Neural Networks*, pages 425–430. Springer, 1998.

[73] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient Non-linear Control Through Neuroevolution. In *Journal of Machine Learning Research JMLR*, volume 9, pages 654–662. Journal of Machine Learning Research JMLR, Berlin, Heidelberg, 9 2006.

[74] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9:937–965, 2008.

[75] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *IJCAI*, volume 99, pages 1356–1361, 1999.

[76] F. J. Gomez and R. Miikkulainen. *Robust Non-linear Control through Neuroevolution.* PhD thesis, The University of Texas at Austin, 2003.

[77] F. J. Gomez and R. Miikkulainen. Active Guidance for a Finless Rocket Using Neuroevolution. In *Genetic and Evolutionary Computation GECCO 2003*, pages 2084–2095. Springer Berlin Heidelberg, Berlin, Heidelberg, 7 2007.

[78] F. J. Gomez and J. Schmidhuber. Co-evolving recurrent neurons learn deep memory pomdps. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 491–498. ACM, 2005.

[79] D. Gravina, A. Liapis, and G. Yannakakis. Surprise search: Beyond objectives and novelty. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 677–684. ACM, 2016.

[80] S. E. Grigorescu, N. Petkov, and P. Kruizinga. Comparison of texture features based on gabor filters. *IEEE Transactions on Image processing*, 11(10):1160–1167, 2002.

[81] F. Gruau. Automatic Definition of Modular Neural Networks. *Adaptive Behavior*, 3:151–183, 1994.

[82] F. Gruau. Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 55–74. IEEE, 2003.

[83] F. Gruau, D. Whitley, and L. Pyeatt. A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks. In *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 81—-89. MIT press, MIT press, 1996.

[84] A. Gruslys, M. G. Azar, M. G. Bellemare, and R. Munos. The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning. *CoRR*, abs/1704.04651, 2017.

[85] V. Gupta, R. Sadana, and S. Moudgil. Image style transfer using convolutional neural networks based on transfer learning. In *International Journal of Computational Systems Engineering*, volume 5, pages 53–60. Inderscience Publishers (IEL), 2019.

[86] P. J. Hancock. Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 108–122. IEEE, 1992.

[87] H. Handels, T. Roß, J. Kreusch, H. H. Wolff, and S. J. Poeppl. Feature selection for optimized skin tumor recognition using genetic algorithms. *Artificial Intelligence in Medicine*, 16:283–297, 1999.

[88] S. Harding and J. F. Miller. Evolution of robot controller using cartesian genetic programming. In *Genetic Programming*, pages 62–73. Springer Berlin Heidelberg, 2005.

[89] I. Harvey. Species adaptation genetic algorithms: A basis for a continuing SAGA. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 346–354. MIT Press, 1992.

[90] I. Harvey. Artificial Evolution for Real Problems. In *Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97). Proceedings of the 5th International Symposium on Evolutionary Robotics, Tokyo*, pages 1–23. AAI Books, 1997.

[91] E. J. Hastings, R. K. Guha, and K. O. Stanley. Evolving content in the galactic arms race video game. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 241–248. IEEE, 2009.

[92] A. T. Hatjimihail. Genetic algorithms-based design and optimization of statistical quality-control procedures. *Clinical Chemistry*, 39:1972–1978, 1993.

[93] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone. HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 217–224. ACM, 2012.

[94] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6:355–366, 2014.

[95] S. Haykin. *Neural networks: a comprehensive foundation.* Prentice Hall PTR, 1994.

[96] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2015.

[97] L. Helms and J. Clune. Improving HybrID: How to best combine indirect and direct encoding in evolutionary algorithms. *PLoS ONE*, 12:1–35, 2017.

[98] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[99] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018.

[100] G. S. Hornby. ALPS: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822. ACM, 2006.

[101] G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial life*, 8:223–246, 2002.

[102] G. S. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita. Evolving robust gaits with AIBO. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 3, pages 3040–3045. IEEE, 2000.

[103] M. Hutter and S. Legg. Fitness uniform optimization. *IEEE Transactions on Evolutionary Computation*, 10:568–589, 2006.

[104] F. N. Iandola, M. W. Moskewicz, S. Karayev, R. B. Girshick, T. Darrell, and K. Keutzer. DenseNet: Implementing Efficient ConvNet Descriptor Pyramids. *CoRR*, abs/1404.1869, 2014.

[105] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Let there be color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classificatio. *ACM Transactions on Graphics*, 35:1–11, 2016.

[106] M. F. Jefferson, N. Pendleton, S. B. Lucas, and M. A. Horan. Comparison of a genetic algorithm neural network with logistic regression for predicting outcome after surgery for patients with nonsmall cell lung carcinoma. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 79:1338–1342, 1997.

[107] B. Jolley and A. Channon. Toward Evolving Robust, Deliberate Motion Planning with HyperNEAT. In *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017 - Proceedings*, volume 2018-January, pages 1–8. IEEE, 2018.

[108] B. Jolley and A. Channon. Evolving Robust, Deliberate Motion Planning With a Shallow Convolutional Neural Network. In *Artificial Life Conference Proceedings*, pages 536–543. MIT Press, 2018.

[109] B. P. Jolley, J. M. Borg, and A. Channon. Analysis of social learning strategies when discovering and maintaining behaviours inaccessible to incremental genetic evolution. In *Lecture Notes in Computer Science*, volume 9825 LNCS, pages 293–304. Springer, 2016.

[110] V. Jurdjevic, J. Velimir, and V. Đurđević. *Geometric control theory.* Cambridge university press, 1997.

[111] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine. Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2017.

[112] S. Kalra, S. Rahnamayan, and K. Deb. Enhancing clearing-based niching method using delaunay triangulation. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2328–2337. IEEE, 2017.

[113] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12:566–580, 1996.

[114] G. M. Khan, J. F. Miller, and D. M. Halliday. Developing neural structure of two agents that play checkers using cartesian genetic programming. In *Proceedings of*

the 10th annual conference companion on Genetic and evolutionary computation, pages 2169–2174. ACM, 2008.

[115] G. M. Khan, J. F. Miller, and D. M. Halliday. Evolution of cartesian genetic programs for development of learning neural architecture. *Evolutionary computation*, 19:469–523, 2011.

[116] M. M. Khan, A. M. Ahmad, G. M. Khan, and J. F. Miller. Fast learning neural networks using cartesian genetic programming. *Neurocomputing*, 121:274–289, 2013.

[117] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[118] S. Kistemaker and S. Whiteson. Critical factors in the performance of novelty search. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 965–972. ACM, ACM, 2011.

[119] H. Kitano. Designing Neural Networks using GAs with Graph Generation System. *Complex Systems*, 4:461–476, 1990.

[120] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe. Planning motions with intentions. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 395–408. ACM, 1994.

[121] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *ImageNet Classification with Deep Convolutional Neural Networks*, pages 1097–1105, 2012.

[122] B. Kwolek. Face detection using convolutional neural networks and gabor filters. In *International Conference on Artificial Neural Networks*, pages 551–556. Springer, 2005.

[123] C. G. Langton. Artificial life: the proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems, held september, 1987. *Los Alamos, New Mexico*, 6, 1989.

[124] C. G. Langton. *Artificial life: An overview.* MIT press, 1997.

[125] J.-C. Latombe. *Robot motion planning.* Springer Science & Business Media, 1991.

[126] S. M. LaValle. *Planning algorithms.* Cambridge university press, 2006.

[127] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8:98–113, 1997.

[128] Y. LeCun and Y. Bengio. Convolutional Networks for Images, Speech, and Time-Series. *The Handbook of Brain Theory and Neural Networks*, 3361:255–258, 1995.

[129] Y. LeCun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, and L. D. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems*, pages 396–404, 1990.

[130] S. Lee, J. Yosinski, K. Glette, H. Lipson, and J. Clune. Evolving gaits for physical robots with the HyperNEAT generative encoding: The benefits of simulation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7835 LNCS, pages 540–549. Springer Berlin Heidelberg, Berlin, Heidelberg, 4 2013.

[131] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life*, 11:329–336, 2008.

[132] J. Lehman and K. O. Stanley. Revising the evolutionary computation abstraction: minimal criteria novelty search. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 103–110. ACM, 2010.

[133] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218. ACM, ACM, 2011.

[134] J. Lehman, J. Clune, D. Misevic, C. Adami, L. Altenberg, J. Beaulieu, P. J. Bentley, S. Bernard, G. Beslon, D. M. Bryson, P. Chrabaszcz, N. Cheney, A. Cully, S. Doncieux, F. C. Dyer, K. O. Ellefsen, R. Feldt, S. Fischer, S. Forrest, A. Frénoy, C. Gagné, L. K. L. Goff, L. M. Grabowski, B. Hodjat, F. Hutter, L. Keller, C. Knibbe, P. Krcah, R. E. Lenski, H. Lipson, R. MacCurdy, C. Maestre, R. Miikkulainen, S. Mitri, D. E. Moriarty, J. Mouret, A. Nguyen, C. Ofria, M. Parizeau, D. P. Parsons, R. T. Pennock, W. F. Punch, T. S. Ray, M. Schoenauer, E. Shulte, K. Sims, K. O. Stanley, F. Taddei, D. Tarapore, S. Thibault, W. Weimer, R. Watson, and J. Yosinksi. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *CoRR*, abs/1803.03453, 2018.

[135] Y. Li, B. Sun, T. Wu, and Y. Wang. Face detection with end-to-end integration of a convnet and a 3d model. In *European Conference on Computer Vision*, pages 420–436. Springer, 2016.

[136] H.-S. Lin, J. Xiao, and Z. Michalewicz. Evolutionary algorithm for path planning in mobile robot environment. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 211–216. IEEE, 1994.

[137] A. Lindenmayer. Mathematical models for cellular interactions in development. *Biol*, 18:300–3, 1968.

[138] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974, 2000.

[139] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep Photo Style Transfer. *CoRR*, abs/1703.07511, 2017.

[140] S. W. Mahfoud. Crossover interactions among niches. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 188–193. IEEE, 1994.

[141] S. W. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana Champaign, 1995.

[142] Martin. Increasing Genomic Complexity by Gene Duplication and the Origin of Vertebrates. *The American Naturalist*, 154:111–128, 2017.

[143] J. Martin, S. N. Sasikumar, T. Everitt, and M. Hutter. Count-Based Exploration in Feature Space for Reinforcement Learning. *CoRR*, abs/1706.08090, 2017.

[144] M. Matarić and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and autonomous systems*, 19:67–83, 1996.

[145] C. Mattiussi. Evolutionary synthesis of analog networks. Technical report, EPFL, 2005.

[146] C. Mattiussi and D. Floreano. Analog genetic encoding for the evolution of circuits and networks. *IEEE Transactions on evolutionary computation*, 11:596–607, 2007.

[147] R. Memmesheimer, I. Mykhalchyshyna, V. Seib, N. Theisen, and D. Paulus. Markerless Visual Robot Programming by Demonstration. *CoRR*, abs/1807.11541, 2018.

[148] M. Meng and X. Yang. A neural network approach to real-time trajectory generation [mobile robots]. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 2, pages 1725–1730. IEEE, 1998.

[149] H. Mengistu, J. Lehman, and J. Clune. Evolvability search: directly selecting for evolvability in order to study and produce it. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 141–148. ACM, 2016.

[150] O. J. Mengshoel and D. E. Goldberg. Probabilistic crowding: Deterministic crowding with probabilisitic replacement. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 409–416. Morgan Kaufmann, 1999.

[151] R. Miikkulainen and K. O. Stanley. Competitive Coevolution through Evolutionary Complexification. *Journal Of Artificial Intelligence Research*, 21:63–100, 2011.

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

[153] J. F. Miller. Gecco 2013 tutorial: Cartesian genetic programming. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 715–740. ACM, 2013.

[154] J. F. Miller, P. Thomson, and T. Fogarty. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. *Genetic algorithms and evolution strategies in engineering and computer science*, 8, 1997.

[155] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the evolutionary design of digital circuits—part ii. *Genetic Programming and Evolvable Machines*, 1: 259–288, 2000.

[156] 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.

[157] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 2015.

[158] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[159] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 762–767, 1989.

[160] D. Moriarty and R. Miikkulainen. Efficient Reinforcement Learning Through Symbiotic Evolution. *Machine Learning*, 22:11–32, 1995.

[161] D. E. Moriarty and R. Miikkulainen. Evolving obstacle avoidance behavior in a robot arm. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 468–475. MIT Press Cambridge, MA, 1996.

[162] D. E. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399, 1997.

[163] D. E. Moriarty and R. Miikkulainen. Hierarchical evolution of neural networks. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 428–433. IEEE, 1998.

[164] G. Morse, S. Risi, C. R. Snyder, and K. O. Stanley. Single-unit pattern generators for quadruped locomotion. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 719–726. ACM, 2013.

[165] J. Mouret and J. Clune. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909, 2015.

[166] J. B. Mouret. Novelty-based multiobjectivization. In *Studies in Computational Intelligence*, volume 341, pages 139–154. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[167] J.-B. Mouret and S. Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *2009 IEEE Congress on Evolutionary Computation*, pages 1161–1168. IEEE, 2009.

[168] J.-B. Mouret and S. Doncieux. Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 627–634. ACM, 2009.

[169] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: an empirical study. *Evolutionary computation*, 20:91–133, 2012.

[170] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver. Massively Parallel Methods for Deep Reinforcement Learning. *CoRR*, abs/1507.04296, 2015.

[171] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih. The Uncertainty Bellman Equation and Exploration. *CoRR*, abs/1709.05380, 2017.

[172] J. K. Olesen, G. N. Yannakakis, and J. Hallam. Real-time challenge balance in an RTS game using rtNEAT. In *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, pages 87–94. IEEE, 2008.

[173] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org, 2017.

[174] A. Owens, P. Isola, J. McDermott, A. Torralba, E. H. Adelson, and W. T. Freeman. Visually Indicated Sounds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2405–2413, 2015.

[175] K. F. Pál. Selection schemes with spatial isolation for genetic optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 169–179. Springer, 1994.

[176] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. van Hasselt, J. Quan, M. Vecerík, M. Hessel, R. Munos, and O. Pietquin. Observe and Look Further: Achieving Consistent Performance on Atari. *CoRR*, abs/1805.11593, 2018.

[177] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza. *A field guide to genetic programming.* Lulu, 2008.

[178] J. K. Pugh, L. B. Soros, P. A. Szerlip, and K. O. Stanley. Confronting the Challenge of Quality Diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 967–974. ACM, 2015.

[179] J. K. Pugh, L. B. Soros, and K. O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[180] L. Qing, W. Gang, Y. Zaiyue, and W. Qiuping. Crowding clustering genetic algorithm for multimodal function optimization. *Applied Soft Computing*, 8:88–95, 2008.

[181] N. J. Radcliffe. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications*, 1:67–90, 1993.

[182] S. S. Raju, P. B. Pati, and A. Ramakrishnan. Text localization and extraction from complex color images. In *International Symposium on Visual Computing*, pages 486–493. Springer, 2005.

[183] N. Richards, D. E. Moriarty, and R. Miikkulainen. Evolving neural networks to play go. *Applied Intelligence*, 8:85–96, 1998.

[184] D. Richmond, A. P.-T. Jost, T. Lambert, J. Waters, and H. Elliott. DeadNet: Identifying Phototoxicity from Label-free Microscopy Images of Cells using Deep ConvNets. *ArXiv*, abs/1701.06109, 2017.

[185] S. Risi and K. O. Stanley. Enhancing ES-HyperNEAT to evolve more complex regular neural networks. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1539–1546. ACM, ACM, 2011.

[186] S. Risi and K. O. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18:331–363, 2012.

[187] S. Risi and K. O. Stanley. A unified approach to evolving plasticity and neural geometry. *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 2012.

[188] S. Risi and K. O. Stanley. Confronting the challenge of learning a flexible neural controller for a diversity of morphologies. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 255–262. ACM, 2013.

[189] S. Risi and J. Togelius. Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9:25–41, 2014.

[190] S. Risi, S. D. Vanderbleek, C. E. Hughes, and K. O. Stanley. How novelty search

escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 153–160. ACM, 2009.

[191] S. Risi, J. Lehman, D. B. D. Ambrosio, and K. O. Stanley. Automatically Categorizing Procedurally Generated Content for Collecting Games. In *Proceedings of the Workshop on Procedural Content Generation in Games (PCG) at the 9th International Conference on the Foundations of Digital Games (FDG-2014).*, 2014.

[192] E. Robinson, T. Ellis, and A. Channon. Neuroevolution of Agents Capable of Reactive and Deliberative Behaviours in Novel and Dynamic Environments. In *Advances in Artificial Life*, pages 345–354, 2007.

[193] T. Salimans and R. Chen. Learning Montezuma's Revenge from a Single Demonstration. *CoRR*, abs/1812.03381, 2018.

[194] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864, 2017.

[195] S. S. Sarwar, P. Panda, and K. Roy. Gabor filter assisted energy efficient fast learning convolutional neural networks. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2017.

[196] G. M. Saunders, P. J. Angeline, and J. B. Pollack. Structural and behavioral evolution of recurrent networks. In *Advances in Neural Information Processing Systems*, pages 88–95, 1994.

[197] J. D. Schaffer. *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1986.

[198] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR).*, pages 1–21, 2015.

[199] J. Schmidhuber, M. Gagliolo, D. Wierstra, and F. Gomez. Recurrent Support Vector Machines. Technical report, Technical Report, no. IDSIA 19-05, 2005.

[200] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. *Neural computation*, 19:757–779, 2007.

[201] M. Schmidt and H. Lipson. Age-fitness pareto optimization. In *Genetic programming theory and practice VIII*, pages 129–146. Springer, 2011.

[202] J. Schrum. Evolving indirectly encoded convolutional neural networks to play tetris with low-level features. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 205–212. ACM, 2018.

[203] J. Secretan and N. Beato. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1759–1768. ACM Press, 2008.

[204] M. Shi. An empirical comparison of evolution and coevolution for designing artificial neural network game players. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 379–386. ACM, 2008.

[205] T. Shibata and T. Fukuda. Intelligent motion planning by genetic algorithm with fuzzy critic. In *Proceedings of 8th IEEE International Symposium on Intelligent Control*, pages 565–570. IEEE, 1993.

[206] B. Sierra and P. Larranaga. Predicting survival in malignant skin melanoma using bayesian networks automatically induced by genetic algorithms. an empirical comparison between different approaches. *Artificial Intelligence in Medicine*, 14: 215–230, 1998.

[207] F. Silva, L. Correia, and A. L. Christensen. R-HybrID: Evolution of Agent Controllers with a Hybridisation of Indirect and Direct Encodings. In *Proceedings of the International Conference on Autonomous and Multiagent Systems*, pages

735–744. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

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

[209] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–9, 2016.

[210] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2:221–248, 1994.

[211] K. Stanley. Exploiting regularity without development. In *Proceedings of the AAAI Fall Symposium on Developmental Systems*, 2006.

[212] K. Stanley, N. Kohl, R. Sherony, and R. Miikkulainen. Neuroevolution of an automobile crash warning system. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1977–1984. ACM, 2005.

[213] K. Stanley, J. Clune, and D. D'Ambrosio. CPPNs Effectively Encode Fracture: A Response to Critical Factors in the Performance of HyperNEAT. *Citeseer*, 2: 1–37, 2013.

[214] K. O. Stanley. Comparing artificial phenotypes with natural biological patterns. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 8–9, 2006.

[215] K. O. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, volume 2, pages 1757–1762. IEEE, 2002.

[216] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 569–577. Morgan Kaufmann Publishers Inc., 2002.

[217] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9:93–130, 2003.

[218] K. O. Stanley and R. Miikkulainen. Evolving a Roving Eye for Go. In *Genetic and Evolutionary Computation Conference*, pages 1226–1238. Springer Berlin Heidelberg, Berlin, Heidelberg, 6 2010.

[219] K. O. Stanley, R. Miikkulainen, S. K.O., M. R., K. O. Stanley, and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[220] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9: 653–668, 2005.

[221] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks. *Artificial life*, 15:185–212, 2009.

[222] A. Stanton and A. Channon. Incremental Neuroevolution of Reactive and Deliberative 3D Agents. In *Proceedings of the European Conference on Artificial Life 13*, pages 341–348. MIT Press, 2015.

[223] A. Stanton and A. Channon. Neuroevolution of Feedback Control for Object Manipulation by 3D Agents. *Artificial Life*, 2016:144–151, 2016.

[224] C. Stanton and J. Clune. Deep Curiosity Search: Intra-Life Exploration Improves

Performance on Challenging Deep Reinforcement Learning Problems. *CoRR*, abs/1806.00553, 2018.

[225] V. Štruc, N. Pavešić, et al. Principal gabor filters for face recognition. In *2009 IEEE 3rd International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–6. IEEE, 2009.

[226] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *CoRR*, abs/1712.06567, 2017.

[227] F. P. Such, V. Madhavan, R. Liu, R. Wang, P. S. Castro, Y. Li, L. Schubert, M. G. Bellemare, J. Clune, and J. Lehman. An Atari Model Zoo for Analyzing, Visualizing, and Comparing Deep Reinforcement Learning Agents. *CoRR*, abs/1812.07069, 2018.

[228] S. Suwajanakorn, S. M. Seitz, and I. Kemelmacher-Shlizerman. Synthesizing obama: learning lip sync from audio. *ACM Transactions on Graphics (TOG)*, 36:95, 2017.

[229] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June-2015:1–9, 2015.

[230] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pages 2753–2762, 2017.

[231] L. Trujillo, G. Olague, E. Lutton, F. Fernández De Vega, L. Dozal, and E. Clemente. Speciation in behavioral space for evolutionary robotics. *Jour-*

nal of Intelligent and Robotic Systems: Theory and Applications, 64:323–351, 2011.

[232] L. Trujillo, L. Muñoz, E. Galván-López, and S. Silva. Neat Genetic Programming: Controlling bloat naturally. *Information Sciences*, 333:21–43, 2016.

[233] Y. R. Tsoy and V. G. Spitsyn. Using genetic algorithm with adaptive mutation mechanism for neural networks design and training. In *Proceedings - 9th Russian-Korean International Symposium on Science and Technology, KORUS-2005*, volume 1, pages 709–714. IEEE, 2005.

[234] A. M. Turing. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52:153–197, 1952.

[235] A. J. Turner and J. F. Miller. Cartesian genetic programming encoded artificial neural networks. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, a comparison using three benchmarks, page 1005. ACM, ACM, 2013.

[236] A. J. Turner and J. F. Miller. Cartesian genetic programming: Why no bloat? In *Genetic Programming*, pages 222–233. Springer Berlin Heidelberg, 2014.

[237] A. J. Turner and J. F. Miller. Neuroevolution: evolving heterogeneous artificial neural networks. *Evolutionary Intelligence*, 7:135–154, 2014.

[238] A. J. Turner and J. F. Miller. Recurrent cartesian genetic programming of artificial neural networks. *Genetic Programming and Evolvable Machines*, 18:185–212, 2017.

[239] J. Vaario, S. Ohsuga, and K. Hori. Connectionist Modeling Using Lindenmayer Systems. In *Information Modeling and Knowledge Bases: Foundations, Theory, and Applications*, pages 496–510. Citeseer, 1991.

[240] P. Vadakkepat, K. C. Tan, and W. Ming-Liang. Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, volume 1, pages 256–263. IEEE, 2000.

[241] R. Vaillant, C. Monrocq, and Y. Le Cun. Original approach for the localisation of objects in images. *IEE Proceedings - Vision, Image, and Signal Processing*, 141:245–250, 2002.

[242] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *CoRR*, abs/1609.03499, 2016.

[243] H. Van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[244] P. Verbancsics and J. Harguess. Image classification using generative neuro evolution for deep learning. In *2015 IEEE winter conference on applications of computer vision*, pages 488–493. IEEE, 2015.

[245] P. Verbancsics and K. O. Stanley. Evolving Static Representations for Task Transfer. *Journal of Machine Learning Research*, 11:1737–1769, 2010.

[246] R. Vierlinger. Towards AI Drawing Agents. In *Modelling Behaviour*, pages 357–369. Springer International Publishing, Cham, 2015.

[247] R. Volpe and P. Khosla. Manipulator control with superquadric artificial potential functions: Theory and experiments. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:1423–1436, 1990.

[248] R. Wang, J. Lehman, J. Clune, and K. O. Stanley. Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions. *CoRR*, abs/1901.01753:1–28, 2019.

[249] Z. Wang, N. de Freitas, and M. Lanctot. Dueling Network Architectures for Deep Reinforcement Learning. *CoRR*, abs/1511.06581, 2015.

[250] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard Universityn, 1974.

[251] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving keepaway soccer players through task decomposition. In *Genetic and Evolutionary Computation Conference*, pages 356–368. Springer, 2003.

[252] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl. Automatic feature selection in neuroevolution. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1225–1232. ACM, 2005.

[253] L. D. Whitley. Fundamental principles of deception in genetic search. In *Foundations of genetic algorithms*, volume 1, pages 221–241. Elsevier, 1991.

[254] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387. IEEE, 2008.

[255] G. Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3:131–150, 1991.

[256] D. G. Wilson, S. Cussat-Blanc, H. Luga, and J. F. Miller. Evolving simple programs for playing atari games. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 229–236. ACM, 2018.

[257] B. G. Woolley and K. O. Stanley. Evolving a single scalable controller for an octopus arm with a variable number of segments. In *International Conference on Parallel Problem Solving from Nature*, pages 270–279. Springer, 2010.

[258] B. G. Woolley and K. O. Stanley. On the deleterious effects of a priori objectives on evolution and representation. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 957–964. ACM, 2011.

[259] K. H. Wu, C. H. Chen, and J. Der Lee. Genetic-based adaptive fuzzy controller for robot path planning. In *Proceedings of IEEE 5th International Fuzzy Systems*, volume 3, pages 1687–1692. IEEE, 1996.

[260] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.

[261] J. Xiao, Z. Michalewicz, and L. Zhang. Evolutionary planner/navigator: Operator performance and self-tuning. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 366–371. IEEE, 1996.

[262] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski. Adaptive evolutionary planner/navigator for mobile robots. *IEEE transactions on evolutionary computation*, 1:18–28, 1997.

[263] L. Xie and A. Yuille. Genetic CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388, 2017.

[264] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87:1423–1447, 1999.

[265] L. Yann, B. Leon, B. Yoshua, and H. Patrick. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

[266] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87:1423–1447, 1999.

[267] C. H. Yong and R. Miikkulainen. Cooperative coevolution of multi-agent systems. Technical report, Department of Computer Sciences, The University of Texas at Austin, 2001.

[268] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[269] B.-T. Zhang and S.-H. Kim. An evolutionary method for active learning of mobile robot path planning. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 312–317. IEEE, 1997.