

THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Evolving Neural Networks for Visual Processing

Oliver Johan Coleman (3275760)

Bachelor of Computer Science (Honours)

Submitted: 19/10/2010

Supervisor: Alan Blair

Assessor: Bill Wilson

Abstract

In this study a recently developed neuroevolution method, HyperNEAT, is applied to new tasks in order to learn more about its characteristics in general and for the tasks specifically. The tasks required directly processing raw image data, including some from a robot-mounted camera, requiring the neural networks to directly process hundreds or thousands of inputs. The tasks included object recognition and robot navigation. The complexity and properties of the tasks, as well as the neural network substrate topologies, were varied and the performance and characteristics of the evolutionary process and solutions generated were analysed. Positive results were achieved for non-trivial tasks, and some important characteristics of HyperNEAT not previously reported were discovered: a bias towards generating weight patterns aligned with the topographical arrangement of neurons within the layers of the substrate network; the impact on performance of the number of dimensions in the topographical arrangement of neurons in hidden layers; the ability to evolve solutions more quickly with larger substrate networks; and a possible difficulty co-evolving weight patterns for substrate networks containing more than one hidden layer. New directions for further research and possible enhancements to HyperNEAT are proposed based on these findings.

Table of Contents

1 Introduction and Problem Definition.....	5
1.1 Background and general motivation.....	5
1.2 Project description and specific motivation.....	6
2 Literature Survey.....	7
2.1 Direct and Indirect Mappings.....	7
2.1.1 Direct mappings.....	7
2.1.2 Indirect mappings.....	7
2.2 Fuzzy Neural Network for Image Classification.....	10
2.3 Grammatical (L-System) Based Approaches.....	10
2.4 Cell-chemistry Based Approaches.....	12
2.5 Hypercube Based Approach.....	15
2.5.1 Overview of hypercube encoding.....	15
2.5.2 Neuroevolution of Augmenting Topologies (NEAT).....	18
2.5.3 Experimental results for hypercube encodings.....	18
2.6 Mechanisms employed by existing encodings.....	23
2.7 No free lunch.....	24
3 Method.....	26
3.1 Kinds of tasks.....	26
3.2 Experiments.....	26
3.2.1 Supervised learning tasks.....	27
3.2.2 Reinforcement learning tasks.....	28
3.3 Implementation and experimental platforms.....	29
4 Supervised learning task – Object Recognition.....	31
4.1 Tasks 1.1 and 1.2 – discrimination between simple solid shapes of different sizes.....	31
4.1.1 Task 1.1.....	31
4.1.2 Task 1.2.....	32
4.1.3 Results for task 1.1 and 1.2.....	32
4.2 Task 1.3 - discrimination between complex non-solid shapes.....	33
4.2.1 Method.....	33
4.2.2 Results and analysis.....	37
4.3 Task 1.4 – rotation and scale-invariant discrimination between complex non-solid shapes.....	39
4.3.1 Method.....	39

4.3.2 Results and analysis.....	40
5 Reinforcement learning tasks.....	49
5.1.1 Technical difficulties.....	49
5.2 Task 2.1.....	50
5.2.1 Method.....	50
5.2.2 Results and analysis.....	51
5.3 Task 2.2.....	54
5.3.1 Method.....	54
5.3.2 Results and analysis.....	55
6 Discussion and Future Work.....	57
7 Conclusion.....	60
8 Appendices.....	61
8.1 Appendix 1 - Parameters for Object Recognition tasks 1 and 2.....	61
8.2 Appendix 2 - Parameters for Object Recognition task 3.....	62
8.3 Appendix 3 - Parameters for Object Recognition task 4.....	63
8.4 Appendix 4 - Parameters for Reinforcement Learning tasks.....	64
9 Bibliography.....	65

1 Introduction and Problem Definition

1.1 Background and general motivation

Artificial neural networks have long held promise in the fields of machine learning and artificial intelligence, if only for the simple reason that we know that human intelligence is based on a neural network substrate. Additionally, neural networks can exhibit some useful properties: they can be applied to many kinds of tasks; be fault tolerant; and generalise well over previously unseen input.

We also know that the human brain has been produced via an evolutionary process. Artificial evolution also has some useful properties: it requires only a scalar evaluation signal, and often performs well over many task domains.

The field of evolution of artificial neural networks - neuroevolution - has also enjoyed success in many kinds of tasks. Most notably it has been successfully applied to tasks where the exact target outputs or policy are unknown, and only a scalar evaluation signal is available. However, until recently neuroevolution methods have typically only been able to produce networks with small numbers of inputs and outputs (low-dimensional input and output spaces); no general method existed for evolving neural networks that can directly process high-dimensional input or output spaces, for example containing hundreds or thousands of inputs or outputs.

In 2007, a method was developed that attempts to overcome this limitation. Hypercube-based Neuro-Evolution of Augmenting Topologies (HyperNEAT) [1] is a neuroevolution method that uses an indirect encoding - hypercube weight pattern encoding - that makes it much easier for the evolutionary algorithm to generate large-scale neural networks by exploiting the geometric regularities in a potentially high-dimensional input space. It has proved successful in evolving neural networks for many kinds of tasks, including: the Robocup Keepaway task (a sub-task of soccer) using a high-dimensional bird's eye view of the field as input [2]; visual discrimination in high resolution input spaces [1]; direct evaluation of board game positions [3]; robot controllers for multi-agent learning [4]; controlling quadruped gaits [5]; and simulated driving [6].

Thus HyperNEAT and its variants are a promising approach, representing a step towards a general, robust method of developing large-scale networks able to process high-dimensional input containing geometric regularities. There are still many open questions and much room for further exploration regarding its behaviour, abilities and variations. Given the initial promising results an important task is to further study the abilities and properties of this method.

1.2 Project description and specific motivation

Given its ability to evolve artificial neural networks (ANNs) for tasks with high-dimensional input containing geometric regularities, HyperNEAT appears to be a promising approach for evolving ANNs for visual processing tasks. It has had some success in this domain, as noted above. However, there are many kinds of visual processing tasks on which it has not previously been tried . In order to add to the knowledge of HyperNEAT's abilities and properties, this study applied HyperNEAT to various visual processing tasks and analysed the results.

This report first presents a survey and discussion of existing neuroevolution methods, HyperNEAT in particular, then discusses the method used to study HyperNEAT. Lastly the details and results of the experiments are presented and discussed.

2 Literature Survey

There have only been two neuroevolution techniques developed that are able to evolve ANNs for the direct processing of high-dimensional input required by visual processing tasks: HyperNEAT [7] and Evolvable Fuzzy Neural Networks (EFUNNs) [8]. The former is a relatively general-purpose approach with numerous published experimental results, whereas the latter is only applicable to image classification tasks and has relatively few published experimental results.

The below literature survey consists of two parts: an overview of neuroevolution techniques and developments to provide context; and a more in-depth description of HyperNEAT and experimental results pertaining to it.

2.1 Direct and Indirect Mappings

There have been many proposed neuroevolution techniques. The two broadest categories distinguish between using a direct or indirect mapping from genotype to phenotype.

2.1.1 Direct mappings

In a direct mapping from genotype to phenotype there is a one-to-one mapping of genes in the genotype to structures in the phenotype. The earliest neuroevolution techniques used this form of mapping. This approach has several limitations:

- the total size and complexity of the evolved network is quite limited: to evolve a large network requires searching through an intractably large genotype space; [9]
- if the same structure is required multiple times in the network, it must be evolved independently as many times as it is required;
- due to these limitations, it is impractical to apply it to tasks with high-dimensional input spaces. For example the second limitation described above would apply for visual input, where the same low-level processing structures are usually repeated over the visual field before higher level processing.

2.1.2 Indirect mappings

The main aim of indirect mappings is to overcome the limitations of direct mappings, namely to decrease the genotype size while evolving relatively large and/or complex phenotypes. This approach is inspired by the process of embryogeny in nature, where a relatively small number of

genes describe bodies and brains with orders of magnitude more structural units.

Indirect mappings achieve compression through gene reuse. Typically this occurs in two ways: a single gene or set of genes can be used to describe a structure (possibly with variation) that repeats in the phenotype, or a gene is used as a trigger for the location of the same or similar structures in some context [9].

Examples of the former in nature are the left-right symmetry of bodies, and the repeating structure of receptive fields in the eye and spine (where the same structure is repeated with some variation). An example of the latter in nature is that of the gene product Fibroblast Growth Factor (FGF), which induces fore limbs or hind limbs depending on where it is applied.

Some indirect mappings for neuroevolution do not explicitly use distinct forms of gene reuse. For example in HyperNEAT, every gene can influence every weight, though this system is capable of evolving both types of gene reuse (symmetry and repetition).

Various authors have used different terms for artificial embryogenic systems, including “artificial ontogeny” [10], “computational embryogeny” [11], “cellular encoding” [12], “morphogenesis” [13] and “artificial embryogeny” [9]. This report adopts the term artificial embryogeny.

Taxonomies of indirect mappings

Traditionally artificial embryogeny (AE) has been further categorised into cell chemistry and grammatical approaches. Cell chemistry describes systems inspired by Turing's reaction-diffusion chemical model [14], where development occurs through localised interactions. Grammatical approaches are systems using or inspired by grammatical rewrite rules such as L-systems. However as noted by Stanley and Miikkulainen [9], all the mechanisms used by both approaches can be combined, and there is a lot of overlap in some existing systems.

For example in [15] grammatical rewrite rules are embedded in a cell chemistry model to describe neuron migration and division, as well as connection growth. Stanley and Miikkulainen [9] seek to propose a more useful taxonomy for AE systems, towards identifying:

- the contribution of specific aspects of development to the capacity to evolve complex systems;
- aspects of biological development that can be implemented efficiently via abstractions on computer systems; and
- gaps in the exploration of AE systems.

In doing so it introduces five characterisations, or aspects, of AE systems:

1. **Cell Fate:** The mechanisms for determining role of a cell at the end of development. For example, location, type of activation function and connections;
2. **Targeting:** The mechanisms for determining the ways that cells create connections to target locations. For example a gene might specify a particular target cell, or the relative direction to the target;
3. **Heterochrony:** How the timing and ordering of developmental events in the embryogeny of a lineage of organisms changes over generations. For example switching the order of development of components can affect the function of the developed system;
4. **Canalisation:** Mechanisms that allow tolerance to mutations. For example allowing developing components to adjust to changes caused by mutations in connected components.
5. **Complexification:** Mechanisms for increasing the size of the genome and resulting complexity of the phenotype over the course of evolution.

For each, they describe numerous corresponding mechanisms in biological evolution and embryogenesis, lending support to the aspects chosen. They note that many of the mechanisms used in biological embryogenesis have yet to be tried in AE systems.

In developing a taxonomy from these aspects for the direct comparison of AE, they define continuous dimensions, or axes, for each, on which an AE system can then be located. For example the dimension for cell fate is defined by the number of mechanisms for cell fate that an AE system employs (at one extreme a system would employ a single mechanism, and at the other extreme a system would employ many). While this makes comparison easier, the justification for the particular methods chosen in the study for simplifying the aspects (as continuous dimensions), or in fact why turning them into continuous dimensions at all is a good idea (other than convenience), is unclear. For example it could be the case that the type of mechanisms used for cell fate are far more important than how many are used.

Another aspect that has been mentioned by several researchers [16],[17] is that of self-repair - the ability of a system that has finished the development stage to repair itself when damage occurs.

Perhaps due to historical reasons most systems designed over the last few decades fall largely into the two categories of cell-chemistry and L-system based. However another kind of indirect encoding, Hypercube encoding [1] has been developed recently that breaks away from these two main approaches. Indirect encodings developed over the last few decades will now be surveyed in

this context, analysed with respect to the five aspects provided in [9] as well as self-repair. The fuzzy neural network approach will also be discussed briefly.

2.2 Fuzzy Neural Network for Image Classification

Kasabov et al. [8] developed an image classification technique based on evolving fuzzy neural networks that took as input the raw image data. Although the method uses a direct encoding it is of interest due to its ability to evolve networks that directly process visual data.

They note that most pattern recognition tasks involve six steps: image segmentation, target identification, attribute selection, sampling, discriminant function generation, and evaluation. Their technique focused on the discriminant function generation task. They applied their technique to two tasks: classification of satellite images, for which they reported good accuracy, and classifying images of fruit damaged by various insects, for which they reported poor accuracy.

2.3 Grammatical (L-System) Based Approaches

At its simplest, a genome in an L-system encoding consists of a set of production, or rewrite, rules. The production rules are evolved. An embryo starts as a single symbol, and the rules are repeatedly applied to the first and subsequent symbols in order to grow the phenotype network. The final resulting symbols are interpreted as a neural network.

Numerous methods based on this approach have been proposed [18-22]. Hornby and Pollack [19] developed a parametric L-system with 20 rewrite rules that was used to evolve the neural networks and body morphologies of robots for locomotion in a simulated 3D environment. They compared this system with a direct encoding of explicit build commands, and found that the developmental system more rapidly evolved fitter robots consisting of more body parts and exhibiting repeating and sometimes symmetrical structures. The results indicated that reuse of structural descriptions in a genome can be a useful part of AE systems, result in better solutions, and accelerate the search through phenotype space - in part by biasing the search in favour of repeating structures.

Luerssen and Powers [23] propose an AE system where the genome consists of rewrite rules that replace labelled hyper-edges in a hyper-graph with sub-hyper-graphs. In a hyper-graph a hyper-edge has arity greater than two. A hyper-edge represents an incomplete part of the phenotype. Once all hyper-edges have been replaced, the resulting graph represents the phenotype. However instead of representing a neural network, the authors envision the nodes being processors that may perform complex operations on their inputs, not just integration. They do not include a scheme for

generating weight values, arguing that back-propagation or competitive learning is better suited to this task and can be integrated into the processors. No experimental results were reported.

Cellular Encoding

Another grammatically inspired approach is that of cellular encoding (CE), which achieves gene reuse via subroutines and recursion, rather than applying rules under certain conditions. Development begins with a single ancestor neuron. Rules described by a grammar tree are applied to the first and then subsequent neurons to develop the network.

Gruau developed the first CE system [24]. The grammar contained instructions for splitting a neuron into two neurons and changing the value of or removing weights. Each new neuron read from its corresponding part of the grammar tree (see Figure 2). The grammar tree was evolved using Genetic Programming (GP). Recursion was achieved by adding a decrementing counter and recursion instruction. This system was able to evolve solutions to the three bit parity task that reused solutions to the XOR task. It also evolved solutions that could solve n-bit parity tasks by reusing the same structure, where the size of the parity problem could be varied by altering the value of a single gene.

Luke and Spector [25] then developed an “edge encoding” system, where networks are grown by modifying edges in a graph instead of nodes, and the grammar tree is traversed depth-first rather than breadth first. The edges in the graph become the nodes in the phenotype network. Their motivation was to address the following problems with CE:

- Crossing over subtrees of CE genomes changes the order in which their operations are executed. Since the effects of CE’s operators depend on their execution order, the same subtree in one genome may result in a very different phenotype when inherited by an offspring.
- Because CE’s operators are node-centric, CE’s ability to make specific and precise modifications to connections is limited.
- CE creates networks by splitting many cells into two or more interconnected cells. Therefore, the networks tend to be highly interconnected, which may slow down the search in some tasks where high connectivity is not required.

Komosinski and Rotaru-Varga [26] found that a CE system similar to Gruau's was able to evolve the bodies and neural network controllers for robots for locomotion in a simulated 3D environment. The solutions typically exhibited repeating structures and in some cases modularity (they surmise

that more modularity may have been exhibited if the task was more complex).

Meyer, Doncieux, Filliat and Guillot [27] developed a geometry oriented variation of CE. In this system the cells occupy a position in a 2D space, and targets for connections are specified by relative coordinates. They also used an incremental training approach. The task was manually divided into sub-tasks: a NN module was evolved for the first sub-task and then “frozen”, and then a new module evolved for the second task, and so on. Neurons in new modules were able to access neurons in existing modules. They were able to evolve controllers for numerous simulated robot tasks.

Arguing that in order to design an efficient neuroevolution method a flexible genetic encoding is needed, Kassahun et al. [28] developed a “common genetic encoding” (CGE). The CGE had the following properties:

- complete: able to represent all types of valid phenotype networks;
- closed: every valid genotype represents a valid phenotype, and genetic operators produce valid genotypes; and
- applicable to both the direct and indirect encodings.

When a genome is interpreted as an indirect encoding, it uses an edge encoding scheme similar to that described by Luke and Spector [25]. While nice to have a common encoding for direct and indirect representations (to which the same genetic operators can be applied), it is not clear that this encoding offers much benefit, as there are several other encodings that have the first two properties (e.g. cellular encoding [24], edge encoding [25] and HyperNEAT [1]). Also, of course, this is just one specific kind of indirect encoding.

2.4 Cell-chemistry Based Approaches

In cell-chemistry based approaches, development is modelled via diffusions of signals and corresponding reactions (reaction-diffusion inspired systems, RD). Typically these systems use some sort of genetic regulatory network (GRN), which is evolved. The GRN creates networks of interactions between the genes in each cell and their environment that lead to a sequence of state changes inside each cell. These systems are usually modelled in a simulated 2D or 3D Cartesian space in which signals can diffuse from cells, cells can move and split and connections can grow. As well as controlling gene expression, the regulatory system may control cell growth and death, movement of cells, growth of connections, and other characteristics of cells such as type of activation function and receptivity to neuro-modulators.

The first RD inspired system was developed by Nolfi and Parisi [29]. Every neuron had a corresponding gene in a fixed length genome, which described the neuron's location and the branching properties of its connections. Connections “diffused”, or grew, out until they hit other neurons. This was not an indirect encoding, but it did allow a simple kind of development stage. In later work Nolfi, Parisi and Cangelosi [30] modified their system to use rewrite rules to describe cell division and migration as well as connection growth. Thus this system used a combination of the grammatical and cell-chemistry approaches.

Next, Dellaert [31], and Dellaert and Beer [32],[33] developed a GRN based system. The GRN was implemented with Boolean functions (“operons”) which when satisfied produce a specific protein. The operons had as arguments the presence (possibly in a specified quantity) or absence of proteins. The proteins could govern many things, including inter-cellular signalling during development, connection growth (connections grew when the right protein was present and would grow around cells according to path-designating proteins, and connect to cells with the right target proteins) and cell division. While biologically plausible, it was found to be too complex to be able to evolve a NN controller for a simple obstacle avoidance vehicle task from scratch; they had to hand-code the genome, after which they could use evolution to fine-tune it.

Later, Dellaert and Beer [34] simplified their system, replacing the operons with Random Boolean Networks (RBN). RBNs, introduced by Kauffman [35], are Boolean expressions whose outputs are connected to the inputs of other expressions, forming a network. The state of a cell was then defined by the output of all the expressions it contained, rather than the quantities of various proteins. They also simplified the connection growth model: a connection was made simply if a cell had the correct target proteins. With this system it was possible to evolve a solution to the obstacle avoidance vehicle task from scratch. In this case a simpler, more abstract (and less biologically plausible) developmental system worked better than the more biologically plausible system.

During this time Jakobi [13] developed another GRN based system. Like Dellaert and Beer's system, proteins interacted with the genome and were also used for inter-cellular signalling. However in this system genes encoded “templates”, which were strings of characters that could be matched with proteins in order to perform protein regulation. The benefit of this system is that the entire genome was just a string, which could be evolved with standard evolutionary algorithms. This system was used to evolve NN controllers for simulated robots that performed obstacle avoidance.

Bongard et al. [36],[10],[37] produced a GRN based system that also used a kind of genetic template matching for proteins. They evolved the body morphology and NN controllers for walking

robots in a simulated 3D environment. The evolved creatures exhibited repeating structures and functional specialisation. Interestingly, the system evolved separate developmental pathways for the body and NN, even though same gene products could affect both NN and body development, thus allowing changes in one aspect without affecting the other.

Astor and Adami [38] developed a GRN based system where the genome is a set of differential equations that take as arguments the cell's internal state and the state of the outside environment. Genes in cells are activated if chemicals that they specify are located within the cell. The cells move and interact through chemical diffusion and axon growth in order to form networks. They demonstrated a hand-coded genome for a neural network that displayed classical conditioning behaviour, i.e. through repeated activation the network learned to associate an output with a stimulus that initially did not activate the output. A distributed genetic algorithm (GA) was also implemented that allowed users to participate in evolutionary experiments, but results were not reported.

Federici [39] introduced a system in which a genotype is itself an NN, which acts as the cell regulatory system. The cell regulatory NN has as input the internal state of the cell and the external state of diffused chemicals in the environment. One of the internal values is the age of the cell. They were able to evolve NN controllers for simulated agents in a simple food gathering task.

Inspired by the coarse-coding methods used in the mammalian visual cortex, Thangavelautham and D'eleuterio [17] developed an ambitious and complex reaction-diffusion GRN-based AE system, called Artificial Neural Tissue (ANT), that uses a coarse-coding framework to enable or disable modular networks. The coarse-coding is implemented by using multiple decision neurons that trigger for certain input patterns and feed their signal into overlapping ranges of signal processing output neurons. The GRN consists of parameters that control cell growth and enable or inhibit parts of the genotype. Each cell contains genes specifying weights, thresholds/biases, choice of activation function and probability ratios for instructing cell division. All parameters are evolved. Gene regulation occurs during the developmental process in addition to when the network interacts with the environment, allowing on-going self-organisation to occur, influenced by the input received. They tested ANT in three different robotic tasks against two direct encodings (a single regular network, and two regular networks arbitrated by a single decision neuron). They report that ANT performed better than the direct encodings. It was able to perform a kind of self-organised task decomposition by generating specialised network modules that are enabled in the right contexts by the decision neurons.

2.5 Hypercube Based Approach

In this section a description and discussion of the hypercube encoding approach is given, then we look at experimental results of this encoding.

2.5.1 Overview of hypercube encoding

HyperNEAT (HN) was first introduced by D'Ambrosio and Stanley [7]. “Hyper” refers to “hypercube based encoding”. HN has several interesting aspects: it is designed to exploit geometric regularity in the input and/or output space; it can evolve very large networks that can process very high-dimensional input [1]; and, while inspired by natural developmental processes, it does not actually use an explicit developmental stage, but instead attempts to abstractly mimic some of the results of natural development, which makes it relatively efficient compared to other (artificial embryogeny) indirect encoding approaches.

A hypercube encoding has two main aspects: the hypercube-based method of mapping from the geometry of the phenotype ANN substrate to its connection pattern, and the method of producing the weight function that performs the mapping. To describe HN we start with the types of functions that perform the mapping, and then describe how they apply in the hypercube encoding scheme.

The weight function (WF), which constitutes the genome, can be any function that takes several inputs and produces a real-valued output. The function could be anything from a lookup table to a program to a composition/network of many types of functions. The latter type of function is used in the original formulation of HN, and is the “NEAT” in “HyperNEAT”. “NEAT” refers to a neuroevolution method using a direct encoding [40] adapted to use many types of activation functions, known as a Compositional Pattern Producing Network (CPPN) [41]. Such a function with two inputs can be used to produce a 2D intensity image. The kinds of images produced typically have natural characteristics, such as repetition, repetition with variation, and symmetry, with some smooth and/or sharp gradients.

This 2D image could then be used to specify the connection weights, or weight pattern, of a single neuron in a feed-forward network with layers consisting of two-dimensional sheets of neurons. The inputs to the function are the coordinates of the source (or target) neurons, and the output is the value of the weight between the source (or target) neuron and the neuron in question.

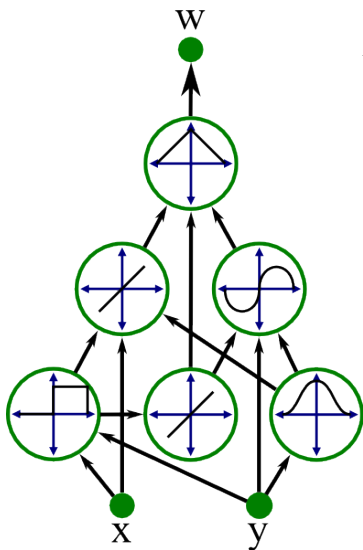


Figure 1: Compositional Pattern Producing Network (CPPN). This could be interpreted as an encoding for a 2D intensity image by plotting the output w as the intensity of a pixel given the coordinates x and y for each pixel.

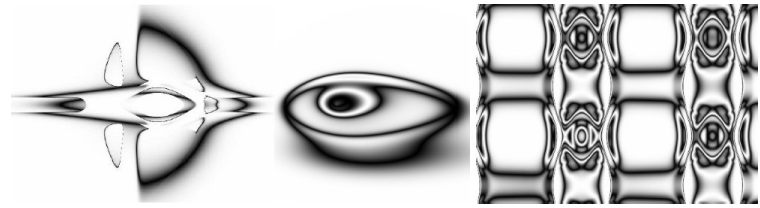


Figure 2: Examples of symmetry, imperfect symmetry and repetition with variation. These images were produced by evolving CPPNs.

HN extends this idea to use a single function to specify the weights of more than one neuron in potentially more than one layer, or in fact in any topology network in a Cartesian space. For example if we wanted the function to specify the weights between just two layers of neurons, we would need 4 inputs: the x and y coordinates for both the source neuron and for the target neuron. Thus the weight pattern of the network is encoded in a hypercube “image”, with the intensities at different points representing the connection weights between different neurons. This can be extended to more than two layers by using another two inputs to the network, being one each for the coordinates of the source and target layer (or just one extra input for a feed-forward network, as the target layer is implicit).

To generate the phenotype the WF is queried for the weight value between every pair of neurons that may have a connection between them (e.g., the topology might be restricted to feed-forward only) by inputting the coordinates of the source and target neurons, and taking the output as the weight value. If the specified weight value is below a threshold, it is not expressed. The dimensions and topology of the network must be specified manually.

Its also possible to use topologies other than grids, and to input geometric information other than coordinates into the WF, such as distance or angle between between neurons. D'Ambrosio and Stanley [7] experimented with using two topologies for a robot food-gathering task, where the robot is circular with range sensors distributed evenly around its body. One topology arranged the input and output layer (there were no hidden neurons) in straight lines, and the other arranged them in concentric rings which more naturally represented the arrangement of the sensors. The inputs to the WF were always the coordinates of the neurons. The experiment found that the linear layout had

slightly better training performance. The authors also tried adding the distance between source and target neurons to the input to the WF; this greatly improved training time and generalisation performance, and also equalised the performance between the two topologies.

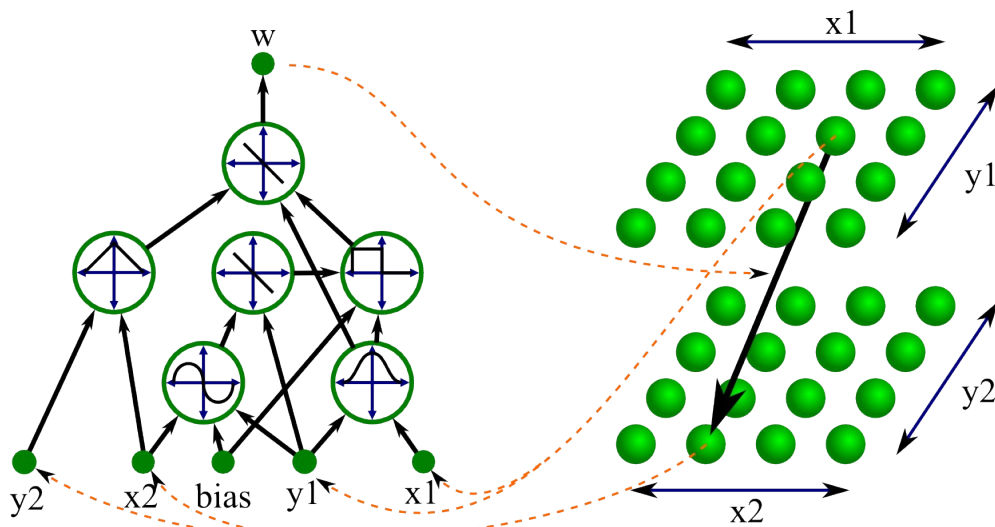


Figure 3: A CPPN interpreted as a weight pattern. The CPPN is queried for the weight of every potential connection between the two layers in the network by inputting the coordinates of the source and target neurons and using the output as the weight value.

The WF can produce functions with repetition, repetition with variation, and symmetry. Thus the weight patterns in the phenotype ANN produced by HN also have these characteristics, allowing repeating structures and modules to be encoded and evolved.

Because HN explicitly encodes the weights of the phenotype NN based on geometrical relationships, it is able to directly exploit the geometrical regularities of the input and/or output space. For example if the input space is a set of range sensors arranged in a regular pattern around a robot, then by arranging them in the same order in the input layer of the network, HN is automatically able to make use of this relationship and any geometric regularities inherent in the kinds of patterns that appear in the input. For example, if detecting walls is relevant to the task, it is relatively easy for the encoding to generate a weight pattern that detects when several neighbouring sensors register something close, and to apply this weight pattern across all sensors. The ability to directly exploit geometric regularity appears to be unique in the field of neuroevolution and opens up a range of tasks that the method may be employed for.

2.5.2 Neuroevolution of Augmenting Topologies (NEAT)

NEAT is a direct-encoding neuroevolution method that has shown strong performance in numerous domains [40],[42],[43]. NEAT is based on three concepts: tracking genetic heritage to allow determining similarity between networks and sensible crossover operators; speciation to maintain genetic diversity and protect new innovations; and complexification by the addition of structural elements over generations. NEAT is used in HyperNEAT to generate weight functions by allowing networks to contain many types of activation functions.

A method is used to keep track of all genes introduced during evolution, with each gene encoding either a connection or neuron. A unique historical marking, or ID, is assigned to every new gene, which are created via structural mutations. This allows determining which individuals are compatible with which, and how their genes can be combined to produce new networks. Genes' IDs are maintained during crossover, so crossover operations can be performed without expensive topological analysis of the networks.

NEAT implements a speciation method to help maintain genetic diversity. Species membership is determined by the number of identical and different genes between the representative of a species and a potential member. A fitness-sharing approach is used where the fitness of an individual is divided by the number of individuals in its species. This allows individuals to compete primarily within their own niches (species) instead of with the entire population, allowing topological innovations to have time to optimise before competing with other niches in the population.

The initial population of networks contain no hidden nodes and differ only in their initial weight values. Over the course of evolution the networks gradually have structure added via mutations, with new innovations being protected using speciation.

2.5.3 Experimental results for hypercube encodings

In [1] Gauci and Stanley introduce HyperNEAT and provide results for it on a simple visual discrimination task: to distinguish between a larger and smaller square in an image. The input and output consisted of grids of neurons of the same size as the image (in pixels). The ANN indicated the centre of the larger square by activating the corresponding output neuron. HyperNEAT was able to evolve NNs for a low resolution version of the task (an image size of 11×11 pixels), and then use the same genome to produce a higher resolution version of the substrate network to process higher resolution images (of size 55×55), without further evolution. The higher resolution networks had over nine million connections.

In [44] Gauci and Stanley argue that making use of the geometric information available for most tasks or domains is critical in enabling machine learning approaches to generalise for those tasks or domains, yet many approaches completely ignore this information. They compared three neuroevolution methods, all based on NEAT, on the game of checkers against a non-trivial deterministic player. The first method represented the board as a single dimension vector, and no geometric information was used; the second method, NEAT-EI, used a hand-engineered topology designed to allow exploiting the regularities inherent in the game (the same topology as Blondie24 [45]); the third method was HN, in which the board position was input directly to a two-dimensional input layer spatially arranged as a grid. The HN implementation utilised a hidden layer the same size as the input layer (8×8), and a single output neuron. They found that NEAT was never able to beat the deterministic player, NEAT-EI did after about 60 generations; and HN did after eight generations. They also found that HN generalised better than NEAT-EI: the most general champions of each run were tested against a slightly randomised version of the deterministic opponent, with HN winning significantly many more games than NEAT-EI.

Further analysis of the weight patterns generated for checkers players [3] found that the ANNs which generalised better also had smoother, more regular weight patterns; that is, thinking of an intensity image representation of the weights from one neuron to the next layer, the images showed smoother, more regular gradients (however still contained some sharp contrasts). They argue that NE methods should encode and represent geometry in a way that creates smooth, regular weight patterns, and that these types of weight patterns more closely resemble biological ANNs.

Woolley and Stanley applied HN to the game of Go [46]. As before, the board position was input directly to a two-dimensional input layer spatially arranged as a grid. However instead of generating a single scalar evaluation for a board position, the network was required to specify where a piece should be placed. The output layer of the network had the same size and spatial arrangement as the input layer, and the output neuron with the highest activation value indicated where the next piece should be placed. The networks were trained against a non-trivial, non-deterministic fixed-policy opponent. They also introduced a method for scaling the size of the substrate neural network during the course of evolution to allow incrementally increasing the board size during training. To achieve this, the initial range of coordinates of neurons in the substrate exist within a sub-space of the hypercube, then as the substrate size increases by adding neurons to the outer edges/surfaces of the substrate, the range of coordinates increases proportionally (thus the existing neurons in the substrate maintain the same coordinates). They achieved positive results against the fixed-policy player while scaling the board size from 5×5 to 7×7 squares.

D'Ambrosio and Stanley [4] extended the concept of representing the geometry of the input space to also represent the geometry of the different roles for agents in a multi-agent task. The same HN genome was used to encode heterogeneous ANNs for all agents. The WF is effectively supplied with extra inputs to specify the coordinates of each agent in a 2D plane, thus allowing it produce similar structures (and policies/behaviours) for all agents, but with variation to allow specialisation. They compared this method against evolving homogeneous ANNs for all agents. They found that for a predator-prey task where the agents are not able to sense each other, the heterogeneous approach performed far better than the homogeneous approach in terms of training times and generalisation. They also tested “seeding” the search for a genome to produce multiple agents by first evolving a single agent to perform the task and then using this genome as the initial genome for evolving a multi-agent genome. In all cases the seeded searches also performed better than unseeded searches in terms of training time and generalisation.

Building on their previous work, D'Ambrosio et al. [47] tested the single WF-heterogeneous multi-agent approach on its ability to scale-up the number of agents for a task without further evolution by interpolating agents in the “policy space” (i.e. coordinates that are fed to the WF to specify the agent's location). They found that for the predator-prey task and a room-clearing task (also where agents are unable to sense each other), they could increase the number of agents from five to 1000 and seven to 23 (and amounts in between) respectively, without adversely affecting performance.

HN has also been successfully applied to a partially observable food gathering task by using a 2D memory grid to create a high-dimensional map of the environment [48]. The substrate networks contained several input layers representing different information (obstacles, navigation history, elevation) at different resolutions, and a single hidden layer. The WF had four inputs (x and y for two layers), and nine outputs, one for each input layer, and one for the hidden layer to output layer.

Drchal et al. [6] applied HN to a simulated driving task. The substrate, including input, hidden and output neurons, were arranged according to polar coordinates, and hence the inputs to the WF were the polar coordinates of the neurons in the substrate.

Clune et al. [49] analysed the sensitivity of HN to different geometric representations of the same problem. The task was to evolve coordinated gaits for a simulated quadruped robot. The inputs to the network were the current angles of each joint, a touch sensor for each leg, and the pitch, roll and yaw of the body. They found that seemingly inconsequential differences in representation produced significantly different performance results. The kind of representation can bias the type of solutions generated.

Buk et al. [50] substituted genetic programming in place of NEAT for the weight function, which they called HyperGP (HG). They found that for a simple simulated driving task both HN and HG gave similar performance results, but that HG exhibited more exploration and more complex solutions in fewer generations.

Togelius et al. [51] applied HyperGP to playing the Super Mario video game. They generated simple recurrent networks (SRN, or Elman networks) that directly processed a high-dimensional visual representation of the game. The agents only performed moderately well, but comparably with other neuroevolution methods tried.

Verbancsics and Stanley [2] applied HyperNEAT to the simulated RoboCup Keepaway task, where a team of agents must keep control of the ball from the opposing team which is trying to take the ball. The input to the ANN was a 20×20 pixel image of an abstract bird's eye view of the field, with scalar values representing the locations of team-mates and opponents. The output is also a 20×20 grid, with the highest value output corresponding to a team-mate nominating where the ball should be passed. They found that HN outperformed the current best reported ML approaches (including TD reinforcement learning and other direct encoding neuroevolution methods). They also reported that using the same genome without further training in variations of the task with more players performed as well as the previous best method of transfer learning applied to the task which required many hours of training during the transfer.

One limitation of the original HN method is that the size and topology of the network must be manually specified. To overcome this limitation Risi et al. [52] propose an extension, called Evolvable Substrate HyperNEAT (ES-HN), to HN that allows evolving the placement and density of neurons in the substrate, based on the premise that information about where to place nodes is implicitly encoded in the weight pattern. The extension is a way of interpreting the output of the weight function to decide where neurons should be placed based on the complexity of the output over a given range in the substrate space, ie where there is more complexity in the weight pattern there are more neurons. The particular method used to determine complexity and specifically where to place nodes effectively defines a language the weight function must learn in order to place nodes. They report positive results for a task that requires hidden nodes. They found that ES-HN had similar training performance to HN even though it also had to evolve the placement of neurons, however it used many more nodes than were required and used in HN for the task.

Woolley and Stanley [46] used HN to evolve controllers for an octopus arm. They argue that the traditional view of the problem domain, where the complexity of the problem is directly related to the dimensionality of the physical domain (i.e. the total number of inputs and outputs to the

controller), obfuscates the underlying concept that needs to be learned. They propose that finding solutions to problems in control should be about discovering an underlying principle and not about the number of dimensions in the action or state representation. They demonstrate that HN evolves solutions that solve the underlying in principle in two ways: by evolving a controller for an eight segment arm (containing 24 muscles) that can then be scaled to a 16 segment, 48 muscle arm and give similar performance without further training; and showing that the training time (measured in evolutionary generations) is unaffected by the number of segments used in the arm (i.e. that the increased dimensionality of a larger arm does not increase the training time).

Risi and Stanley [53] introduced three approaches to evolving adaptive networks, whose connection weight values change online as a result of the activation levels of the source and target neurons of a connection. The approach was to extend HN to generate patterns of learning rules as well as patterns of (initial) weight values. The three methods are, in order of most general and computationally expensive to least general and computationally cheap: (1) add three inputs to the WF representing presynaptic activity, postsynaptic activity and current weight value, then update the substrate every simulation step by re-querying the WF for every connection in the substrate, with the presynaptic activity, postsynaptic activity and current weight value inputs being set to the corresponding values for each weight; (2) a method where the WF has four additional outputs describing parameters for a Hebbian plasticity rule, whose values can be different for every substrate connection; and (3) a method where the WF has one additional output describing the single parameter for a simple Hebbian plasticity rule whose only adjustable parameter is the learning rate (which again can be different for every connection). They applied the three methods to a T-maze problem under two scenarios: one where the reward switches location about halfway through a set of trials; and one where the reward structure is non-linear. They found that there is a trade-off between the generality of an indirect encoding of plasticity and its computational cost, and that even with a substrate topology with no hidden layers it was possible to evolve learning rules with the most general approach to solve the non-linear task.

Interestingly, all the experiments which used substrates consisting of more than two layers used the same encoding method in the WF: the WF still only had four inputs to specify x and y (or polar) coordinates for the source and target neurons, but multiple outputs to specify the weight values between two or more combinations of layers. It is not clear whether other methods, for example using an input to the WF to specify layer coordinate(s), were tried and did not perform as well or if using multiple outputs has become the de facto standard.

2.6 Mechanisms employed by existing encodings

Nearly all systems are using the same few kinds of mechanisms to achieve some or all of the aspects of an AE system. Table 1 briefly outlines the mechanisms used by the various types, or traditional categories, of neuroevolution methods.

	<i>L-systems</i>	<i>Cellular encoding</i>	<i>Reaction-diffusion/GRN</i>	<i>Hypercube</i>
<i>Cell fate</i>	<p>Prepatterning. Most systems use this; the fate of a new structural unit is determined solely by its parent.</p> <p>Content sensitivity. Some systems employ context sensitive rules.</p>	<p>Prepatterning.</p>	<p>Prepatterning. ie. the cells lineage via cell-splitting.</p> <p>Signalling. Signals from other cells or the environment can influence any property or behaviour of a cell. This can allow for chain-reactions and self-organisation.</p>	<p>None.</p>
<i>Targeting</i>	<p>Specific. Connections are explicitly specified.</p> <p>Offset. Connections are specified by their offset in the rewrite string.</p>	<p>Specific.</p> <p>Offset.</p>	<p>Relative. Connections grow out at a specific angle and sometimes distance, or according to some chemical gradient.</p> <p>Semi-specific. Connections are made to cells expressing a matching gene.</p> <p>Combinations of the above are also used.</p>	<p>Geometry. Geometrical relationships between neuron locations.</p>
<i>Heterochrony</i>	<p>None.</p> <p>Parameterisation. Allows expression of rules to change over time.</p>	<p>Parameterisation.</p>	<p>Signalling. Signals (eg gene expressions) can change over time inside and outside cells.</p> <p>Age parameter in cell.</p>	<p>None.</p>
<i>Canalisation</i>	<p>None.</p>	<p>Imprecise targeting. A connection may be expressed even if the targeting is not exact (for cells located in a plane or space).</p>	<p>Imprecise targeting.</p> <p>Apoptosis. Planned cell death, eg if too many cells are produced for a component then some can be removed or disabled during development.</p> <p>Self-organisation. Some systems have a high-degree of self-organisation during development (eg via signalling).</p>	<p>Shared genes. All genes can influence all connections and structures, so a change in one part of the network can be reflected in others. Note that this will not necessarily happen in a beneficial way.</p>

<i>Comp lexification</i>	<i>Variable-length genome.</i>	<i>Variable-length genome. Specialised crossover operators.</i> Do not “break” genomes with crossovers that do not make sense (more likely to occur in larger genomes).	<i>Variable-length genome. Specialised crossover operators.</i>	In terms of the complexity of the weight pattern, rather than the number of neurons and connections, this is inherited from the weight function.
<i>Self-repair</i>	<i>None.</i>	<i>None.</i>	<i>Self-organisation. Redundancy.</i>	<i>None.</i>

Table 1: Mechanisms used by existing types of artificial embryogeny systems.

2.7 No free lunch

Hornby and Pollack [19], who developed an AE based on L-systems, and Gruau and Komosinski et al. [54],[26], who developed CE systems, reported that evolving solutions with direct encodings produced solutions of lower fitness than solutions generated via indirect encodings for the same tasks. They all conclude that their AE system biased the search towards repeating structures, and that this accelerated the search and produced fitter solutions for the tasks they tested on. However, the obvious corollary to this is that the search is necessarily less biased towards finding solutions to tasks that do not require repeating structures.

Stanley and Miikkulainen [40] found that a direct encoding was able to evolve solutions to a non-Markovian version of the problem of simultaneously balancing two poles on a cart using 25 times fewer evaluations than CE. In [9] they conclude that this indicates that the capacity to reuse genes does not ensure efficiency. However it's not clear why gene reuse should help in this task. A more useful interpretation might be that for tasks that do not require repeating structures, an encoding that is biased towards repeating structures may hinder performance.

While not evolving neural networks, Roggen and Federici [16] compared a direct encoding against two different developmental encodings, one a simple reaction-diffusion model and the other the GRN model using a NN as the cell regulatory system developed previously by Federici [39]. The task was to reproduce two images with different types of regularity. They found that for low resolution images the direct encoding performed best, but as the resolution increased the developmental encodings came to perform better.

Clune et al. [55] compared the performance of a direct and indirect encoding (NEAT and HyperNEAT) on tasks where the regularity was incrementally decreased in various ways from highly regular to random. They found that in all cases the indirect encoding outperformed the direct encoding when the problem was highly regular, but at some point as the regularity was decreased the direct encoding would come to outperform the indirect encoding.

These results support the idea that there is a direct trade-off between the performance for evolving solutions with repeating structures, modularity and/or symmetry and evolving solutions without these.

3 Method

The goal of this study is to add to the knowledge of HyperNEAT's abilities and properties. This was achieved by applying HyperNEAT to various visual processing tasks, over varying degrees of complexity and types of regularity, and analysing the results.

3.1 Kinds of tasks

There are many kinds of tasks and variations of tasks that could be tried. These will be outlined broadly and then a list of specific tasks and variations will be presented.

Two common learning paradigms have been considered:

Supervised training - tasks for which a specific output pattern is required given some input pattern, such as classification/pattern recognition or feature extraction, for example extracting roads from satellite images.

There are existing methods for generating ANNs in a supervised learning context, eg back-propagation, however it is useful to see if HyperNEAT is applicable to these kinds of tasks, as it would provide an entirely new approach for tackling this kind of problem. The only reported result on this kind of task with HyperNEAT is the visual discrimination task [1].

Reinforcement learning - tasks where the optimal output is not known given some input pattern and instead only a scalar evaluation signal is available (either at the end of a trial or during the course of a trial), for example robot control where it is not clear how camera images should be processed or what the exact sequence of motor commands should be.

Reinforcement learning tasks are interesting because no existing ANN methods have been successfully applied to this kind of task in the visual processing domain (without significant pre-processing or high level features of the visual input being supplied manually by the practitioner).

3.2 Experiments

Given the time frame available it was not possible to test HyperNEAT on a long list of tasks. The approach taken was to test HyperNEAT on two kinds of tasks, one a supervised learning task and one a reinforcement learning task. For each kind of task the complexity and types of regularity were varied over several sub-tasks in order to see how the solutions and performance of HyperNEAT change. In some tasks the topology of the substrate network was also varied in order to study how this affects the solutions HyperNEAT generates.

The general approach was to select initial tasks that extended on previous reported experiments for the reason that it is known that HyperNEAT works for these tasks and so we could be assured of getting some positive results. The later, more complex tasks were intentionally chosen so that it was not known in advance whether they would be solvable by HyperNEAT so as to provide information about potential limitations of HyperNEAT.

An overview of the experiments is given below, as well as a description of the software used, then the next section provides details of each experiment and results.

3.2.1 Supervised learning tasks

Object recognition and localisation - In a previous proof-of-concept experiment [1] the task was to distinguish between a larger and smaller square that are randomly located in a visual field (image). The input and output consisted of 2D grids of neurons of the same resolution as the image. The task required setting the output corresponding to the centre of the larger square to the highest value among all outputs. This task is a very simple object recognition and localisation task, where it is assumed the target is in the image and that there is only one other non-target object in the image. Note that this task only requires determining the input region with the highest value density, and that the task can be solved by an ANN with no hidden neurons, simplifying the search space.

The experiments performed for this project, designed to incrementally increase the complexity of the task towards more general object recognition and localisation, were:

1. Reproduce previous visual discrimination experiment with larger and smaller square.
2. Same as previous but add multiple smaller squares. Note that this task still only requires determining the input region with highest value density, but there is less difference between the target region and background.
3. Same as previous but use complex, non-solid shapes, such as X and circles and irregular randomly generated shapes. The target shape stays constant for the run. This task is designed to test the ability to recognise a shape rather than the region of highest value density.
4. Discriminate between target and non-target shapes that are randomly rotated and/or scaled between trials (but not specify the location of the target). In this task only the target or a non-target shape are in the image, and the ANN has only output to specify which is being shown. This reduced the size of the ANNs required and so allowed for quicker simulation and evolutionary runs (which given the limited time available was a significant issue).

3.2.2 Reinforcement learning tasks

Robot navigation and control - A birds-eye-view/map image has been used in a previous robot navigation experiment [48]. Multiple input layers were used for different elements of the map and at different scales of the map. The task was to explore an area, avoid obstacles and find food which could not be seen (thus the basic strategy was to visit regions not previously visited). The Robocup Keepaway task has also been successfully tackled using a birds-eye-view representation of the field, but only using a high-level, abstract control method (ie not directly controlling the speed and direction of the robots) [2].

While it would be possible to experiment with a similar task to those mentioned above using a birds-eye-view, a more interesting experiment, simply because it has not yet been tried with HN, would be to create a task using images from a robot-mounted camera. At its simplest this could require the robot to move towards a single object in an empty arena, requiring the ANN controller to turn the robot on the spot when the target is not in view, and steer and move towards it when it is in view. Given the results of previous experiments, such as the visual discrimination task and robot navigation tasks, it seems very likely that this is a feasible task. It also expands on the previous object recognition tasks, and provides a lot of opportunity for complexification and variation and therefore exploring the abilities of HyperNEAT in the reinforcement learning paradigm.

The experiments, again designed to incrementally increase the complexity of the task to a point where it is was not clear that HyperNEAT would work, are:

1. Find and move towards a stationary white sphere in an empty arena. This simply required turning on the spot until the target was in view and then moving approximately directly towards it.
2. Same as previous but the target and robot were hidden in or behind a simple maze constructed of grey walls, requiring an exploration strategy (other than turning on the spot). The maze was designed to be solvable by a reactive agent.

Experiments for these tasks were performed using the SimBad [56] robot simulator (modified to run with no GUI). Camera images are generated using a 3D rendering library (Java3D) which takes advantage of hardware graphics acceleration using OpenGL.

3.3 Implementation and experimental platforms

There are several open-source implementations of HyperNEAT and NEAT available. The preferred language of the author is Java¹, for which there was no HyperNEAT implementation available but several implementations of NEAT. Thus one of the NEAT implementations was extended to implement HyperNEAT. The NEAT software package used is called ANJI (Another NEAT Java Implementation) [57]. ANJI was chosen as it seemed the best engineered and most easily extensible package of the three Java implementations.

After adding code to implement the hypercube encoding scheme and modifying ANJI to be able to use multiple types of activation functions to produce the CPPN, it was found that ANJI contained several bugs and significant differences to the original NEAT algorithm. These were fixed or modified to work like the original algorithm:

- Added more parameters to control speciation, including: minimum species size to select elites² from; minimum number of elites to select; and target number of species (this is controlled by adjusting the compatibility threshold between species).
- In the original NEAT algorithm there is a parameter to specify the percentage of individuals used as parents to produce the next generation but that do not necessarily become part of the next generation. In the ANJI implementation this was confused with elitism such that it determined the number of individuals that would survive intact to the next generation.
- In the original NEAT algorithm there is a parameter to specify the maximum number of generations a species can survive without improvement in its fitness value (after which all the individuals in it will be removed and not selected for reproduction). This was added to ANJI, but in experiments was found to hinder performance (for a wide variety of values).
- In the two existing HyperNEAT implementations re-speciation (discarding existing species groupings and creating new ones with new representative individuals) is forced every so often (based on the amount of change in complexity, measured by genome length, of the population over time). This was approximated by forcing respeciation every 20 generations

1 For several years after its introduction Java was not suited for high performance computing applications due to its being semi-interpreted instead of producing native machine code. However the advent of runtime environments including Just In Time (JIT) compilers has largely overcome this issue, and numerous benchmarks have found it comparable in (time) performance with natively compiled code.

2 Elites are the fittest individuals within a species, and survive intact (without mutations) from one generation to the next.

for some experiments (the object recognition (OR) experiments). Preliminary experiments found this boosted performance (versus no forced respeciation) for the OR experiments but not the robot navigation experiments.

- There was a bug where an individual could be removed from the population list but not from the relevant species member list when the population size was being adjusted after reproduction due to rounding errors. This caused problems when determining the size and average or total fitness of a species.
- There was a bug where elites could be removed from the population when the population size was being adjusted.

An efficient (in time and memory usage) layer/grid-type neural network simulator has also been implemented, with the following features:

- Network can consist of multiple layers of neurons arranged in 2D grids.
- Supports feed-forward and recurrent topologies (however only feed-forward was used for experiments performed in this project).
- Allows specifying the maximum distance connections can be made for neurons in each layer (rather than assuming a fully connected topology between each layer).
- Allows the layers to be of different sizes.

Finally, care was taken to maximise the efficiency and concurrency of the most frequently executed code. This was achieved by:

- Reusing the substrate data structures (instead of recreating new objects and arrays for every evaluation or generation);
- making the code for transcription from a CPPN function to substrate network weight values, substrate network simulation, and evaluation functions (including 3D robot simulation), which accounted for approximately 99 percent of the computation time, multi-threaded to take advantage of all available processors/cores; and
- keeping the frequently used data structures (eg the neural network substrate weight values and activation levels) as small as possible to allow fitting inside the processors cache (or as much as possible) to minimise main memory accesses.

The software developed will be made available on a website dedicated to HyperNEAT research.³

³ <http://eplex.cs.ucf.edu/hyperNEATpage/HyperNEAT.html>

4 Supervised learning task – Object Recognition

These tasks required discriminating between a target and non-target object in an image. The first three tasks also required specifying the location of the target within the image.

Grey-scale images were used in all experiments. The details of the experiments are presented, then the results presented and discussed. The sub-tasks are referred to as task 1.x, where x is the number of the sub-task.

4.1 Tasks 1.1 and 1.2 – discrimination between simple solid shapes of different sizes

The parameters used for these sub-tasks are listed in Appendix 1 - Parameters for Object Recognition tasks 1 and 2.

4.1.1 Task 1.1

The first sub-task was to reproduce the visual discrimination experiment in [1]. In this task the ANN is presented with a visual field (image) in which a large and small square are randomly positioned (not overlapping), and the NN output corresponding to the centre of the large square must be set to the highest value.

The field size was 11 by 11 pixels, so the substrate consisted of an input and output layer consisting of 121 neurons each, and 14641 connections between them (fully connected). The output layer activation function was Sigmoid. The squares were represented as value 1 and the background as value 0.

The large square was 3 by 3 pixels and the small square 1 pixel. Evaluations consisted of 75 trials, with both squares randomly positioned for every trial. The fitness function used was the distance between the centre of the largest square and the location of the output with the highest value, subtracted from maximum possible distance and averaged over the 75 trials.

In this task experiments were run that provided the CPPN with and without x and y delta values (which directly provide the CPPN with information about the distance between source and target neurons in the x and y axis).

4.1.2 Task 1.2

The second sub-task is the same as the first except multiple small squares are present. In the first experiment 20 small squares are randomly positioned (with no overlapping), and in the second 40 small squares were positioned. When the number of small squares exceeds 40, they would start to frequently form the same size square as the large square by chance, making the task impossible. In this task the CPPN was always provided with delta values.

4.1.3 Results for task 1.1 and 1.2

Performance versus generation number is plotted for tasks 1.1 and 1.2 in Figure 4. Results are averaged over 25 runs.

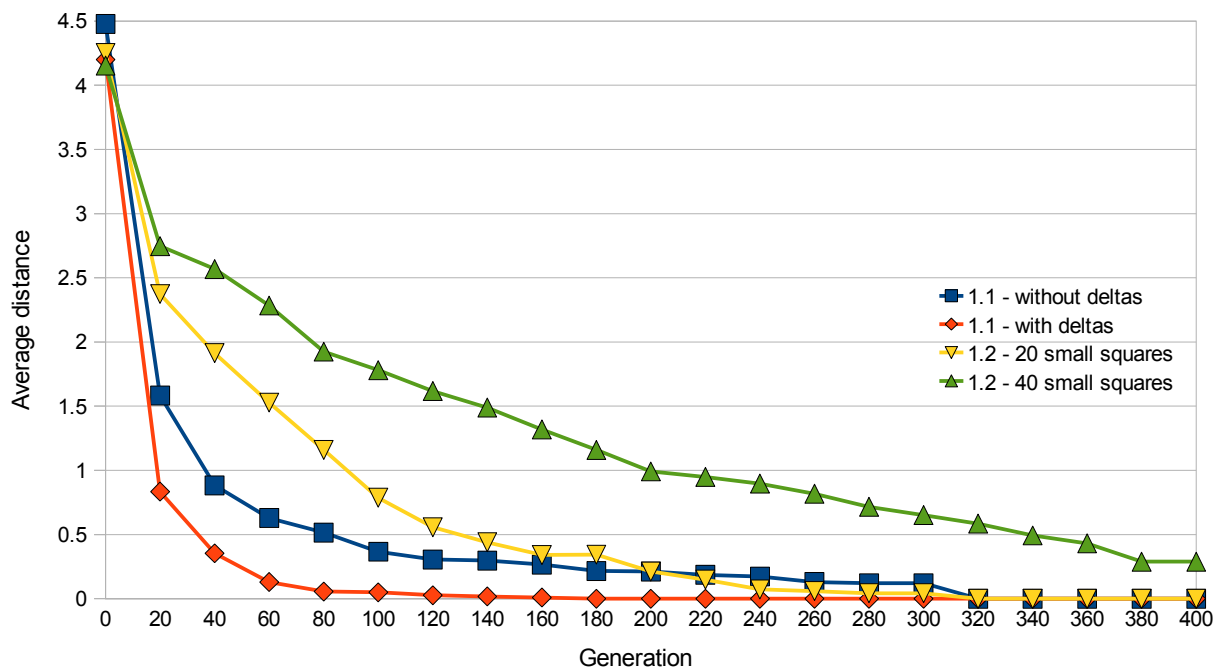


Figure 4: Performance results for tasks 1.1 and 1.2, averaged over 25 runs. Performance is measured by average distance from the centre of the large square to the output neuron with highest activation. For task 1.1 performance is shown for the case when the CPPN is provided with the x and y delta values (“with deltas”) and when it is not (“without deltas”). For task 1.2 performance is shown for when there are 20 and 40 small squares (the CPPN is provided with deltas in both cases). For the case of 40 small squares runs consisted of up to 400 generations.

For task 1.1 results similar to the original experiment in [1] were obtained (both reach an average distance accuracy of about 0.1 by generation 300). However it was found that providing the CPPN with delta values provided more of a boost than in the original experiments. In the original experiment it was found delta values gave an initial boost but the non-delta performance caught up

with it by generation 70. In our experiment the non-delta version does not catch up even after 300 generations. This could be attributable to the slightly different ways the squares are randomly placed and/or to the different function sets available to the CPPN.

For task 1.2 convergence on a good solution took longer in proportion to the number of small squares used. However, even when 40 small squares are used, so that trials frequently contained squares the size of the large square but missing 1 or 2 pixels, HyperNEAT still found good solutions. Note that the solution to this task is the same as that for task 1.1: locating the centre of the input region with the highest density of high values. Adding more smaller squares simply increases the noise and the probability of having regions of similar density to the large square (although the large square is also more likely to have multiple small squares adjacent to it, increasing its relative density).

4.2 Task 1.3 - discrimination between complex non-solid shapes

This task extends on the previous task by using complex non-solid shapes, instead of solid squares, for the target and other objects. As before, the network must specify the location of the target in the image by setting the output neuron corresponding to the centre of the target shape to a higher value than any other output. The parameters used for this task are specified in Appendix 2 - Parameters for Object Recognition task 3.

4.2.1 Method

For each generation of the evolutionary process a set of 200 trial images was generated, against which each individual was tested. Each trial image was generated by randomly placing the target shape in the image and then randomly placing a non-target shape in a position that does not overlap (however it was possible for the corners of the bounding-boxes of the shapes to partially overlap). All shapes were positioned so that they did not extend beyond the edge of the image.

The following shape libraries and target shapes were used (all shapes are drawn as outlines, not solid, and unless otherwise specified all shapes within a library had the same bounding-box dimensions):

- ***square-x-circle***: the target shape was a square, non-target shapes were an X and a circle.
- ***circle-square-x***: the target shape was a circle, and non-target shapes were an X and a square.
- ***x-circle-square***: the target shape was an X, non-target shapes were a circle and a square.
- ***Cs***: this library contained four simple C shapes, which were constructed of three straight

lines, each rotated 90 degrees relative to the previous. The target shape for a run was chosen randomly.

- **random-vh:** shapes in this library were constructed of four randomly placed vertical and/or horizontal lines of random length. For each run 100 shapes were generated and the target selected randomly.
- **random:** shapes in this library were constructed of four randomly placed lines of random length and orientation. For each run 100 shapes were generated and the target selected randomly.

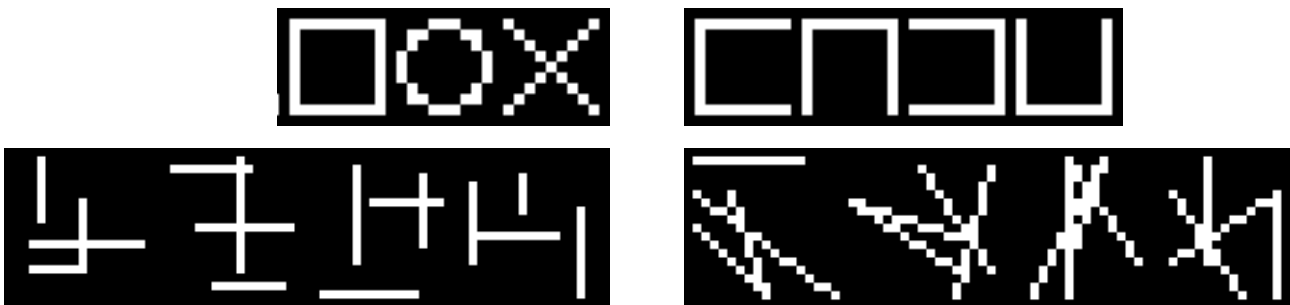


Figure 5: Examples of shapes from the various shape libraries. Top left: square, circle and X shapes, top-right: Cs, bottom-left: random-vh (four examples shown), bottom-right: random (four examples shown).

The fitness function used was a combination of a weighted sum of squared error (WSSSE) and percentage of trials correct (PC). Each part, WSSSE and PC, contributed half each to the total fitness value.

The WSSSE function was calculated by determining the squared error for each output neuron, and then giving the error for the correct target neuron a weight of 0.5 and the weight of all the other error values combined a weight of 0.5. This function provided a smooth gradient to push evolution in the “right general direction”.

The PC function was determined by the percentage of trials for which the output corresponding to the centre of the target shape had the highest value, providing a bias towards accurate solutions.

This combination of functions was found experimentally to produce the best performance. Combinations using functions based on the distance or inverse of the distance from the highest output to the centre of the target were also tried.

Substrate inter-layer connection range/length limiting

The range/length of connections in the substrate network (ie the maximum distance in the x or y axes a connection can extend from a neuron in the input layer to a neuron in the output layer) was limited to the maximum size of the shapes (see figure 6). This resulted in there being far fewer connections in the substrate network, making simulation much faster. It also makes the task of evolving the connection weights easier as the weight pattern only needs to be specified for the relevant part of the image/visual field for each output neuron.

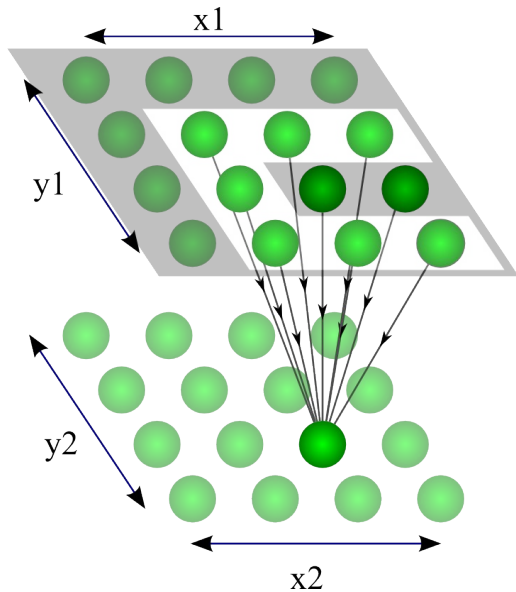


Figure 6: A substrate network that has a limited connection range. The connections to a single output neuron are shown. The maximum connection range is 1. Neurons from the input layer (top), are limited to connecting to neurons in the output layer (bottom) at most 1 neuron away in either the x or y axes. This would correspond to a shape size of 3×3 pixels. An example image containing a C shape is shown, with the shape located in the region covered by the output neuron for which the connections are shown.

Incremental substrate resolution scaling during training

In a previous experiment the ability to increase the resolution of the substrate network without further evolution was demonstrated.[1] To scale the resolution of the substrate the CPPN is queried at a higher resolution (another way to think about this is that the weight pattern hypercube still has the same dimensions, eg unit-size, but it is rendered at a higher resolution).

Scaling of the substrate resolution was used in these experiments to enable quicker evolution while still producing a final network that works accurately for high resolution images. Initially the image (and therefore substrate network input and output)

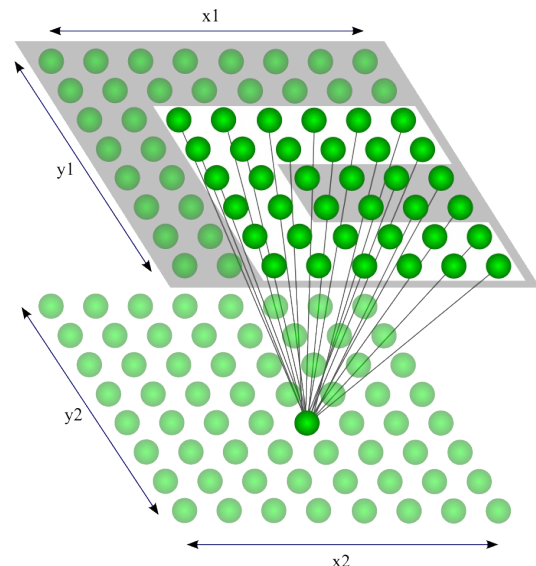


Figure 7: Substrate resolution scaling. This image shows the same substrate shown in figure 6 but at double the resolution.

resolution was 15×15 pixels with shape size 5×5 pixels. When a predetermined fitness value (corresponding to about 97% accuracy) was reached during evolution the substrate network and image resolution were approximately doubled and evolution continued. After the first scaling the resolution of the network and shape were 27×27 and 9×9 respectively, and after the second and final scaling 51×51 and 17×17 respectively. The maximum connection range was increased in step with the shape resolution. The initial resolution of 15×15 pixels resulted in a substrate network with 4986 connections, which took about 0.00025 seconds to simulate. For the final resolution of 51×51 the substrate network contained 634626 connections, which took about 0.034 seconds to simulate (136 times longer than for the initial resolution).

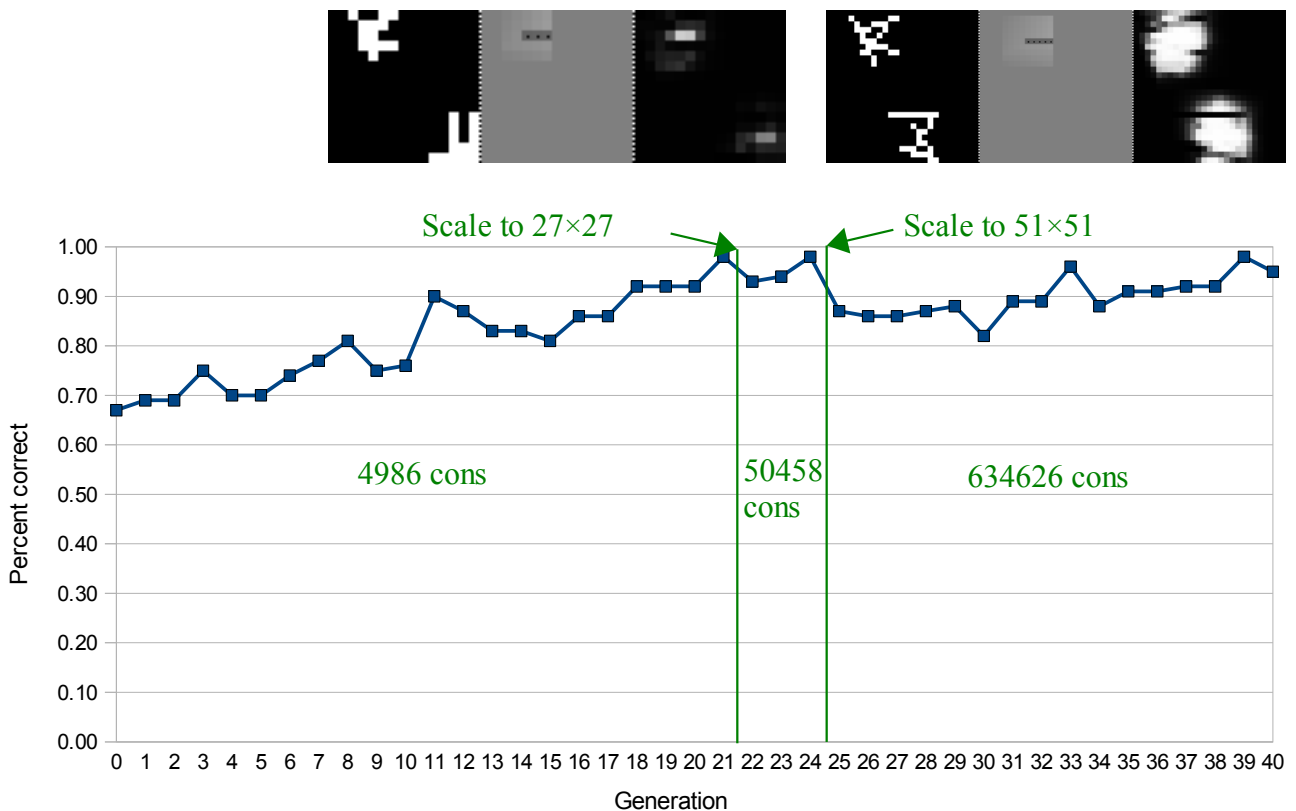


Figure 8: Plot of performance from a single run using substrate resolution scaling. The first scaling occurs at generation 21 and performance drops to 93% but recovers in three generations. The second scale then occurs at generation 24 and performance drops to about 86% and recovers after 15 generations. Performing 40 generations at the final resolution of 51×51 would have taken about 8400 seconds, but this run took about 3400 seconds, or 40% of the time. The top two trial images show the scaling of the input, weight pattern and output resolution, from 27×27 to 51×51 pixels. The left part of the image shows the input to the network, the middle part shows the weight values from all input neurons to the single output neuron corresponding to the centre of the target shape, and the right part shows the output of the network.

4.2.2 Results and analysis

The performance results for each shape library are plotted in Figure 9. The results show the percentage of trials for which the correct output was given (ie the output neuron with highest value was the neuron corresponding to the centre of the target shape), for the best individual for each generation. The results are averaged over 30 runs. Due to the use of resolution scaling and averaging over 30 runs the plots only show the performance from a run after the last resolution scaling for that run (the generations at which scaling occurs differs for each run; and it does not make sense to average over results from different resolutions). This means that the performance from a run is recorded as zero until the generation that the final scaling is performed.

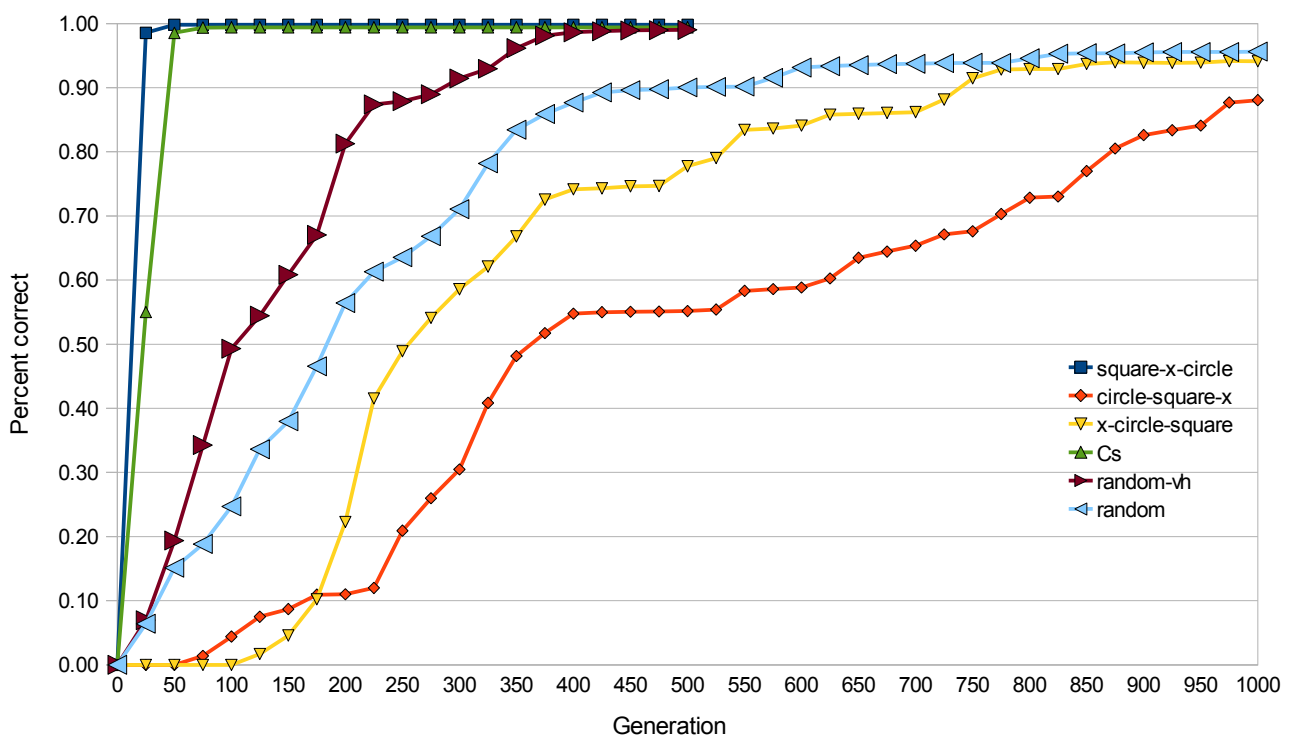


Figure 9: Performance for each shape library over at most 1000 generations, averaged over 30 runs. Surprisingly some of the apparently simplest shape libraries seemed to be the most difficult for HyperNEAT to solve, in particular for the circle and X target shapes. The results indicate that NEAT was biased towards generating weight patterns containing horizontal and/or vertical patterns.

This task required evolving a weight pattern for all output neurons that is similar to the pattern of the target shape. The degree of similarity will also depend on how similar the target shape is to the non-target shapes.

Figure 10 shows four trial images from a run using the *Cs* library. The target shape is the backward

facing 'C'. The left part of a trial image shows the input to the network, the middle part shows the weight values from all input neurons to the single output neuron corresponding to the centre of the target shape, and the right part shows the output of the network.

In the four images the weight pattern is the same (or very similar) for the four output neurons that are in different positions, indicating that the CPPN is reproducing the same pattern translated across the substrate. The weight pattern is similar enough to the target shape that it produces its maximum output only for this shape (although by looking at the output images it is hard to tell if the right output neuron is really the one with the highest output; it only has to be slightly higher than any other for the trial result to be correct).



Figure 10: Trial images for a run using the Cs library. The left part of a trial image shows the input to the network, the middle part shows the weight values from all input neurons to the single output neuron corresponding to the centre of the target shape, and the right part shows the output of the network (50% grey represents zero weight, black represents a maximum negative weight value, white represents a maximum positive weight value, a black dot indicates a negative value). The same weight pattern is reproduced over four output neurons in different positions. The weight pattern produces maximal output for the target shape only.

The order in which HyperNEAT solved this task for different shape libraries, in order of most quickly to least quickly, is: *square-x-circle*, *Cs*, *random-vh*, *random*, *x-square-circle*, and then *circle-square-x*. In the first three shape libraries the target contains only horizontal and vertical (HV) lines. In the fourth (*random*) the target contains lines that are at random angles but often contains some approximately HV lines, in the fifth the target (an X) contains only two lines at 45 degree angles, and in the sixth the target (a circle) contains curved lines. Thus there appears to be a correlation between the presence of HV lines in the target shapes (and possibly between straight versus curved lines) and the ease with which HyperNEAT solved the task. It does not seem to matter if the non-target shapes also contain HV lines or not.

Initially the CPPN inputs were the coordinates and delta (distance) in the x and y axes between the source and target neurons. With only these inputs HyperNEAT had even more difficulty evolving solutions for the *circle-square-x*, *x-square-circle* and *random* libraries. After adding an input representing the angle of the line joining the source and target neurons (relative to the x -axis in the

substrate) performance was improved for these libraries but still lagged behind that for libraries where the target shape consisted only of HV lines (these are the results shown in the performance plot).

This indicates that HyperNEAT, given the CPPN inputs used for these experiments, was biased towards evolving weight patterns aligned with the axis of the grids of neurons in the substrate (ie the x and y axis). This is no surprise given that all seven CPPN inputs (excluding the bias input) related to these axes: the x and y coordinates for the source and target neurons, the delta values (distance between source and target neurons in x and y axis), and the angle between the x axis and the line joining the source and target neurons (although as noted this reduced the bias). Various combinations of activation functions were made available to the CPPN but this did not significantly affect the result.

4.3 Task 1.4 – rotation and scale-invariant discrimination between complex non-solid shapes

This task extends on the previous task by randomly rotating or scaling the shapes within the image. However this task dropped the requirement to specify the location of the target in the image, only requiring discriminating between a target and non-target shapes. This change was made to allow completing experiments in a practical time frame.

The input to the network was the raw image and the networks had a single output to specify whether the shape in the image is the target or a non-target shape. The output encoding was for the target the output value should be greater than 0.5 and for a non-target less than 0.5.

The parameters used for this task are specified in Appendix 3 - Parameters for Object Recognition task 4.

4.3.1 Method

For each generation of the evolutionary process a set of 200 trial images was generated, against which each individual was tested. Half of the trial images contained the target shape and half contained a non-target shape. Each trial image was generated by randomly rotating and/or scaling the shape and placing it in the centre of the image. The shapes were generated at the start of a run and were of the *random* or *random-vh* type used in the previous task.

The fitness function used was the root mean squared error (over the 200 trials), with error calculated assuming the ideal output value is 1 for the target and 0 for the non-target.

No incremental substrate scaling was used and the substrate network topology was fully-connected between all layers (there was no connection length limiting as used in the previous task).

The image size for all experiments was 21×21 pixels (and hence this was the size of the input layer of the network).

The following parameters were used for scaling and rotation of the shapes:

- Scaling: the shape size varies between 11 and 21 pixels and there were 49 non-target shapes, both *random* and *random-vh* shapes were used;
- Rotation: the shape size is 15 pixels, shapes are rotated between 0 and 360 degrees, there were 4 non-target shapes, and only *random-vh* shapes were used.

4.3.2 Results and analysis

The results for scale, rotation and scale and rotation-invariance are shown in Figures 11, 14 and [X] respectively. The results show the percentage of trials for which the correct output was given (that is, the output had value greater than 0.5 if the target shape is present or less than 0.5 otherwise), for the best individual for each generation. The results are averaged over 30 runs. Note that evolution was stopped once over 98% accuracy was reached (percentage of trials correct), so this was approximately the maximum accuracy achievable by any run. This was more an oversight than by design, however by the time it was discovered it was too late to re-run the experiments.

Scale-invariance

For the scale-invariant object recognition task the substrate networks had a single hidden layer of size 6×6 . HyperNEAT achieved the maximum accuracy (98%) after about 150 generations on average for both shape types.

This result appears to be in contrast to the previous (non-scale-invariant object recognition) task where it took longer to evolve solutions for the random shape type than the random-vh shape type. It was hypothesised that this was the case because HyperNEAT was biased towards creating weight patterns containing vertically or horizontally aligned elements. This hypothesis is consistent with all shape types used in that experiment as well as the typical weight patterns produced in the rotation- and scale-invariant object recognition tasks (see below). Thus it is suggested that for the scale-invariant task there is some element to the task that mitigates the effect on performance of the bias towards generating weight patterns aligned with the axes of the spatial arrangement of neurons within the layers. This may be because a single horizontal or vertical line is generally not enough to

recognise the target shape at different scales, so whether or not the target contains only horizontal or vertical lines the weight patterns must contain elements that are not so aligned.

Figure 12 shows six examples of trial images and resulting activation levels for each layer in the substrate networks for a typical solution. Activation levels for the input layer (image) are shown at top, the hidden layer in the middle and the output neuron at the bottom. The left two trials contain the target shape and the right four contain non-target shapes.

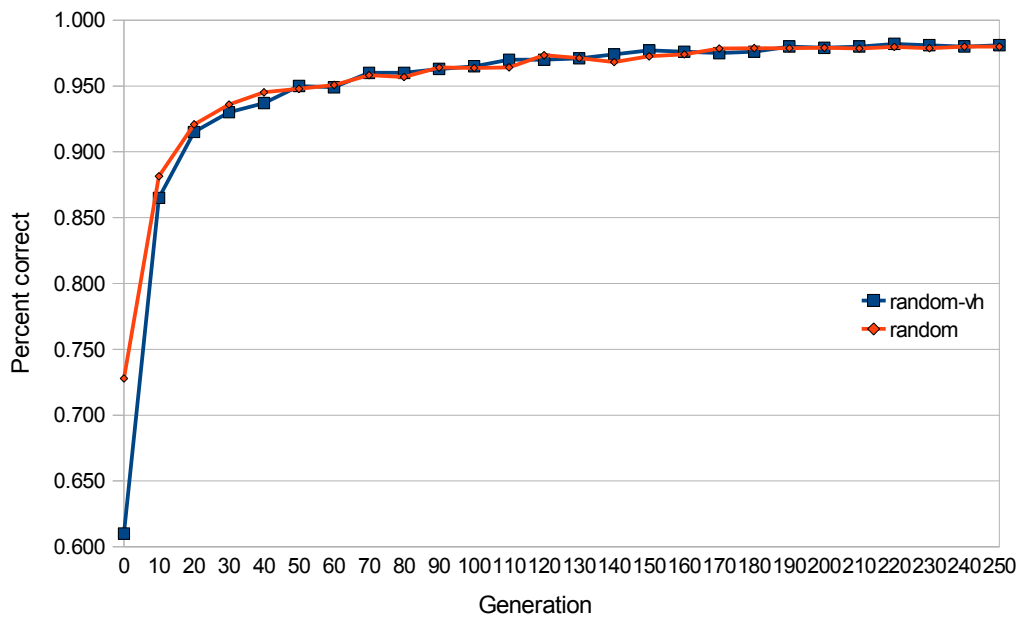


Figure 11: Accuracy results for scale-invariant object recognition over 250 generations, for both the random and random-vh shape types, averaged over 30 runs. The maximum achievable accuracy (98%) is reached after about 150 generations for both shape types.

The hidden layer implements feature detectors to detect the presence/absence of target or non-target elements at different scales. The encoding of the feature detectors is such that they create high activation levels when non-target elements are in the image and low activation levels when target-like elements are in the image.

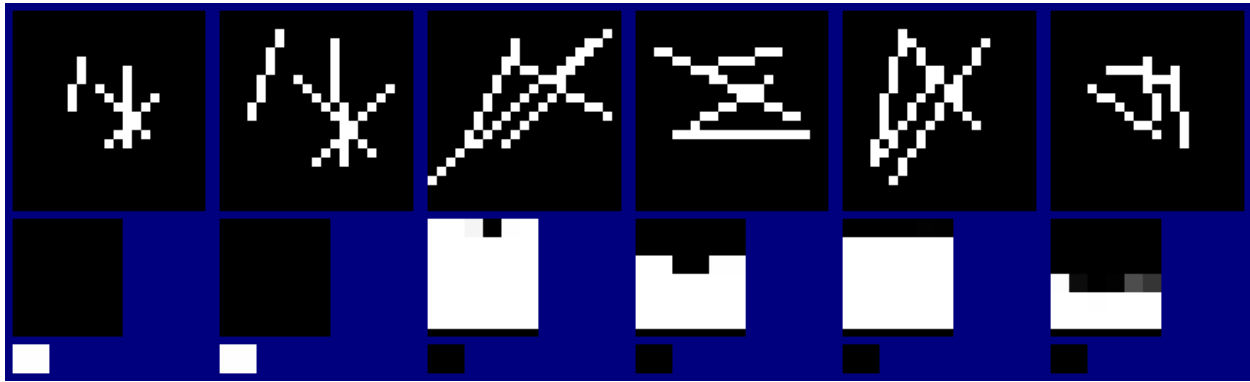


Figure 12: Examples of trial images and resulting neuron activation levels for each layer in the substrate network for the scale-invariant object recognition task (using the random shape type). Six trials are shown. The input layer/image is shown at top, the hidden layer in the middle and the output layer (a single neuron) at bottom. The left two trials contain the target shape and the right four contain a non-target shape. The hidden layer appears to implement feature detectors for non-target elements at different scales.

The activation levels in the hidden layer seem to be spatially correlated, with neighbouring neurons detecting similar features. Since HyperNEAT encodes the weight patterns of the substrate network as a function of the spatial information of the source and target neurons this is what we would expect to see.

Figure 13 shows the weight patterns generating the activation levels in the trial images shown in figure 12. The top array of images show the incoming (from the input layer) weight values for each of the 36 neurons in the hidden layer. The bottom image shows the incoming (from the hidden layer) weight values for the output neuron. Note that the bias weight values are not shown.

The incoming connection weight patterns for each neuron in the hidden layer vary smoothly over the two spatial dimensions of the layers (explaining the spatially correlated activation levels shown in figure 12). As one moves along the x axis from neuron to neighbouring neuron the incoming connection weight patterns for each neuron vary in one way, and as one moves along the y axis the patterns vary in another way.

The positive weight values (white) detect the presence of non-target elements in the image, and the negative weight values (black) detect the presence of target-like elements. The variation in the patterns allows detecting the same elements at different scales, and possibly different elements of the shapes. The incoming connection weight patterns for the output neuron are all negative, and essentially detect if any of the hidden layer neurons are activated. We can infer that the output neuron has a positive bias so that it is activated if none of the hidden layer neurons are.

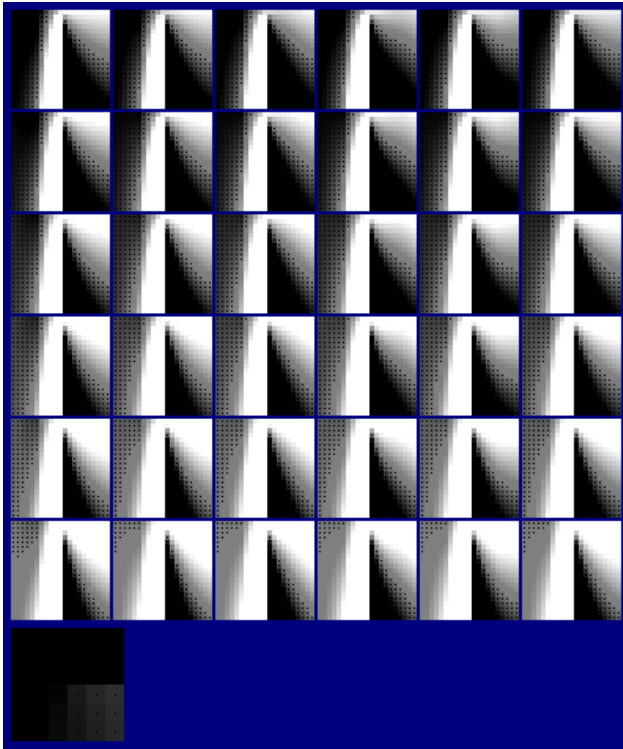


Figure 13: Weight patterns for a typical solution for the scale-invariant object recognition task (using the random shape type). The top array (of size 6×6) of images show the incoming (from the input layer) weight values for each of the 36 neurons in the hidden layer. The bottom image shows the incoming (from the hidden layer) weight values for the output neuron. 50% grey represents a weight value of zero, lighter a positive value and darker a negative value (with a black dot in the centre). Note that the bias weight values are not shown.

In this example, the hidden neurons only become activated when a non-target shape is present. In other solutions some of the hidden neurons would become activated when the target is present and others when a non-target is present. In these solutions it was still the case that the hidden layer weight patterns varied smoothly over the two spatial dimensions of the layers. The output layer weight pattern also varied smoothly from positive values on one side or corner of the layer to the other side or corner, depending on whether the corresponding hidden neurons detected the presence of the target or non-target elements. This demonstrates that HyperNEAT was able to co-evolve the weight patterns for two layers of connections.

Rotation-invariance

For the rotation-invariant object recognition task various substrate network topologies were tried. Four single hidden layer topologies were tried, with hidden layer sizes of 6×6 , 11×11 , 21×21 and 41×41 . Two double hidden layer topologies were tried, with the size of the first hidden layer being 21×21 and the size of the second hidden layer being 6×6 or 11×11 . Figure 14 shows the accuracy achieved with each topology.

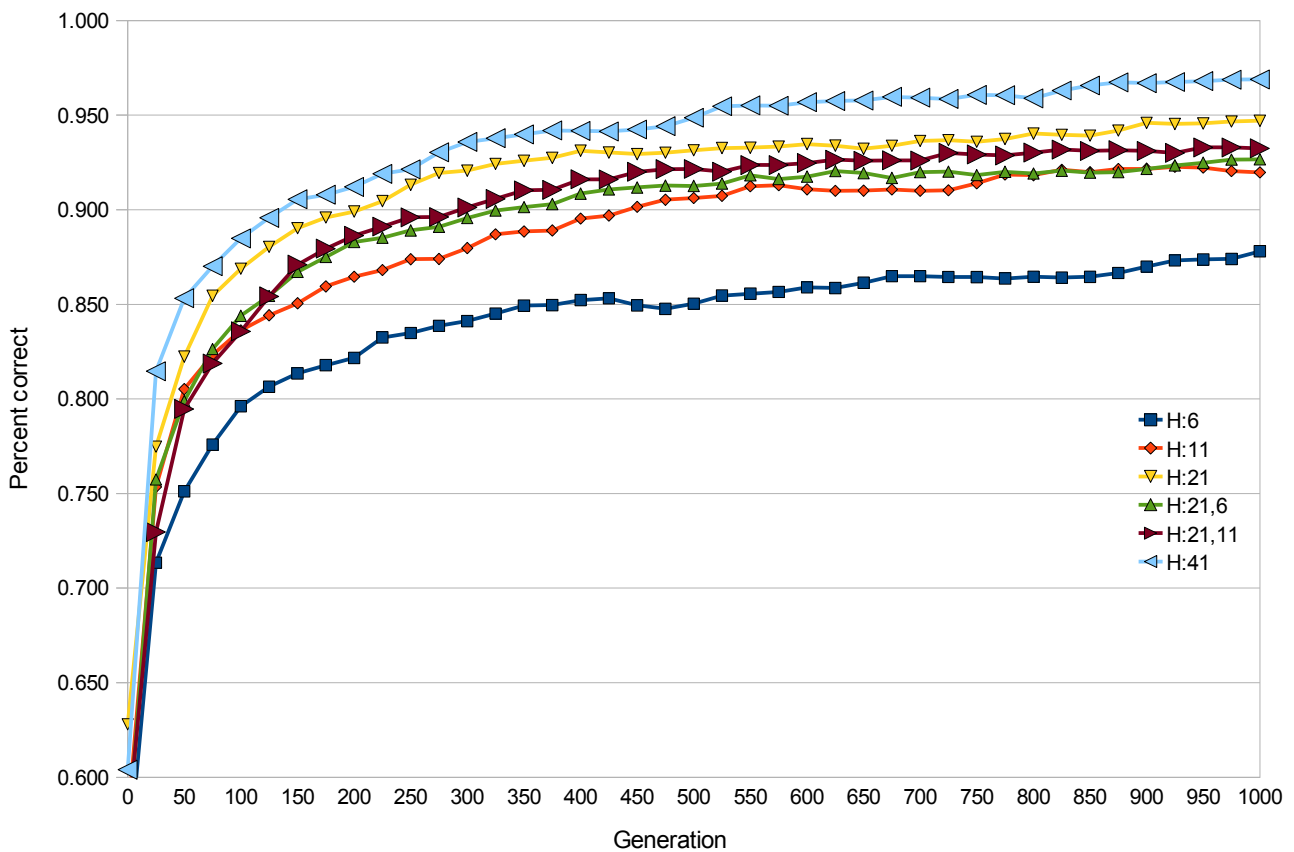


Figure 14: Accuracy results for rotation-invariant object recognition for 6 different substrate topologies over 1000 generations. The topologies varied by the number and size of the hidden layers. Each hidden layer had the same number of neurons in the x and y dimensions (they were square). The legend describes the number and size of hidden layers for each topology in the format $\langle \text{hidden layer 1 size} \rangle$ [, $\langle \text{hidden layer 2 size} \rangle$]. Thus the smallest single hidden layer network had a hidden layer of size 6×6 and the smallest double hidden layer network had hidden layers of size 21×21 and 6×6 respectively. The maximum achievable accuracy (98%) is almost reached (to within 1%) with the topology with a single hidden layer of size 41×41 after about 900 generations. Multiple smaller hidden layers do not perform as well as a single larger hidden layer.

The accuracy achieved with each topology appears to be correlated with the size of the largest hidden layer. The performance for the four single hidden layer topologies increases as the size of the hidden layer increases, with the maximum achievable accuracy (98%) being reached to within 1% with a hidden layer of size 41×41 after about 900 generations. During evolution the accuracy achieved with the three smaller single hidden layer topologies tends to plateau at about 500 generations, however for the next 500 generations (after which evolution was stopped) the accuracy continues to slowly increase. It is not clear whether the accuracy would reach that obtained with the

larger topologies if evolution continued for enough generations. However it is clear that HyperNEAT was able to utilise the larger number of neurons to achieve higher accuracy more quickly: for example, on average the topology with a hidden layer of size 41×41 achieves 90% accuracy by generation 140, whereas on average the topology with a hidden layer of size 11×11 achieves 90% accuracy by generation 450.

Having two hidden layers did not increase the accuracy achieved, as compared to a single hidden layer. The topology with a single hidden layer of size 21×21 achieves a slightly higher accuracy (though not significantly) than the double hidden layer topologies with hidden layers of size 21×21 for the first layer and 6×6 or 11×11 for the second layer.

Optimising weight patterns for multiple hidden layers is more difficult than doing so for a single hidden layer, as the multiple layers of connections must be co-evolved. However, since the representational power of the multiple hidden layer networks is greater than that of the single hidden layer network, we may also expect to see the accuracy of the multiple hidden layer networks continue to increase above the level achieved by the single hidden layer networks. In this case they plateau at the 600th generation at an accuracy slightly lower than that achieved with the single hidden layer topology and do not increase after another 400 generations. It is possible that after enough generations HyperNEAT could make use of the greater representational power afforded by the two hidden layers but there is nothing in these results to suggest that this is the case. It is also possible, however seems less likely, that a network with two hidden layers can not perform better than that with a single hidden layer for this task.

Thus it appears that HyperNEAT, at least for this task and with the parameters and settings used, was unable to optimise weight patterns for two hidden layers as well as those for a single hidden layer. One explanation for this may be that a “competing conventions” problem tended to emerge, with the three layers of connections processing their input according to differing conventions. This is not surprising given that to successfully co-evolve the connection weights over multiple layers of connections a change in one layer will generally require updating the weights (to match the updated “convention” of the changed layer) in the other layers. HyperNEAT was clearly able to do this reliably for two layers of connections (a single hidden neuron layer) as shown in both this and the scale-invariance task, but appears to start struggling to do this with more than two layers of connections (two hidden neuron layers).

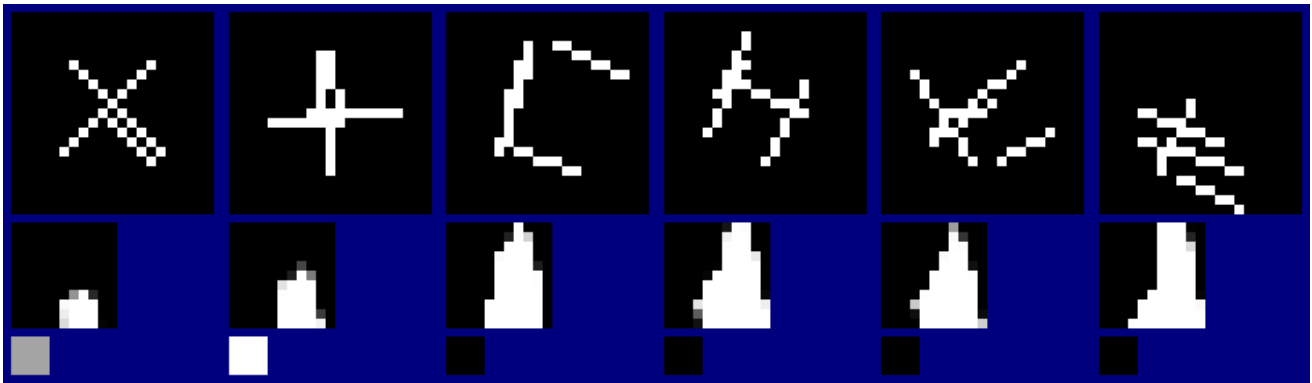


Figure 15: Examples of trial images and resulting neuron activation levels for each layer in the substrate network for the rotation-invariant object recognition task. Six trials are shown. The input layer/image is shown at top, the hidden layer in the middle and the output layer (a single neuron) at bottom. The left two trials contain the target shape and the right four contain a non-target shape. The hidden layer appears to implement feature detectors for target and non-target elements at different orientations.

Figure 15 shows six examples of trial images and resulting activation levels for each layer in the substrate networks for the rotation-invariant object recognition task. The left two trials contain the target shape and the right four contain non-target shapes.

Similarly to the hidden layers for the scale-invariant task, the activation levels in the hidden layer seem to be spatially correlated, with neighbouring neurons detecting similar features.

Figure 16 shows the weight patterns generating the activation levels in the example trial images shown in figure 15. The top array of images show the incoming weight values for each of the 121 neurons in the hidden layer. The bottom image shows the incoming weight values for the output neuron.

Again similarly to the weight patterns for the scale-invariant task, the incoming weight patterns over all the neurons in the hidden layer vary smoothly over the two spatial dimensions of the layers. In this solution some of feature detectors in the hidden layer detect (becoming activated) when target-like elements are present in the image, while others detect the presence of non-target elements. The weight values for the incoming connections for the output neuron vary in accordance with this, again demonstrating that HyperNEAT was able to co-evolve the weight patterns for two layers of connections, but this time with a more complex pattern in the second layer of connections.

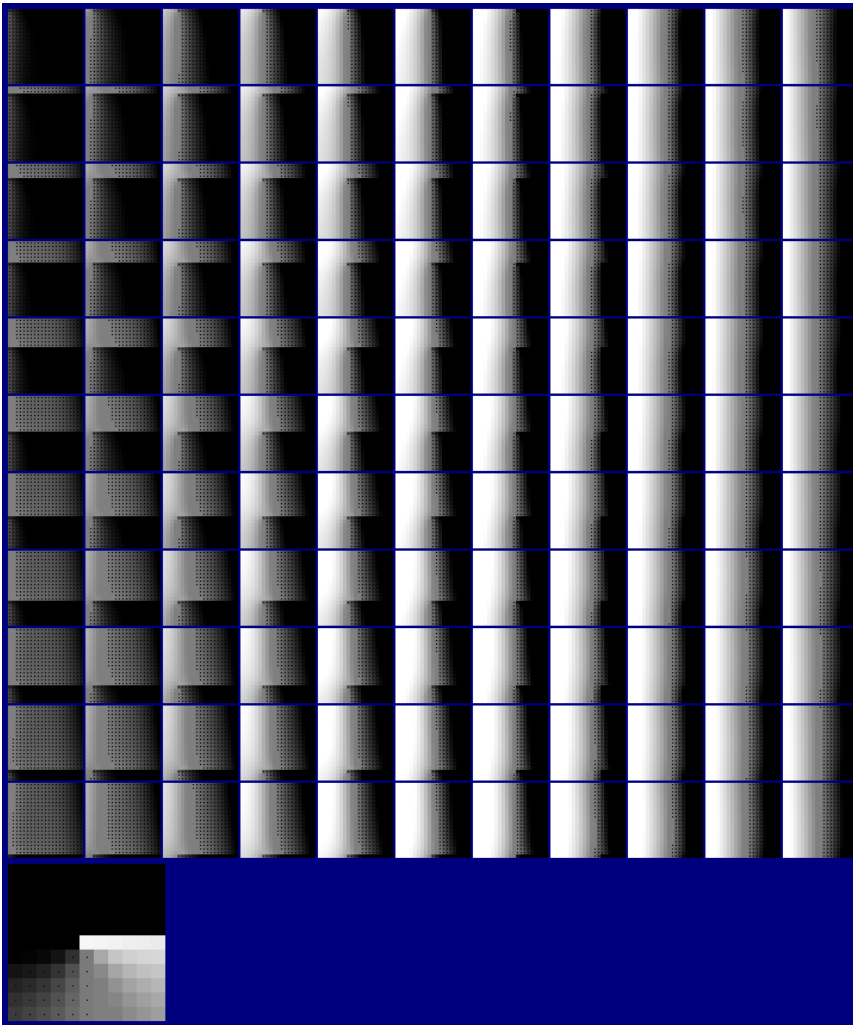


Figure 16: Weight patterns for a solution for the rotation-invariant object recognition task, for the topology with a single hidden layer of size 11×11 . The top array of images show the incoming weight values for each of the 121 neurons in the hidden layer. The bottom image shows the incoming weight values for the output neuron.

One-dimensional versus two-dimensional spatial arrangement of neurons in the hidden layer

Figure 17 shows the accuracy achieved (measured by percentage of trials correct) when the hidden layer is reduced to one spatial dimension, that is, a single line of neurons instead of a two-dimensional plane of neurons. These networks had the same number of neurons in the hidden-layer as the two-dimensional equivalents, and the same connection topology (fully connected between layers). Thus the representational power of the networks is unaffected and only the spatial organisation of the neurons in the hidden layer as perceived by the hypercube encoding scheme is changed.

The two hidden layer sizes that experiments were run for, by total number of neurons, was 121 and 441, corresponding to layer sizes of 11×11 and the equivalent 121×1 , and 21×21 and the equivalent 441×1 .

The performance for the one-dimensional spatial arrangement is lower than the equivalent two-dimensional arrangement for the duration of evolution (500 generations) for both the larger and smaller hidden layer sizes. For the 121 neuron layer size accuracy at generation 100 is about 9%

lower for the one-dimensional arrangement versus the two-dimensional arrangement but almost catches up towards the end. For the 441 neuron layer size accuracy from generation 100 until evolution was stopped is about 9% lower for the one-dimensional arrangement versus the two-dimensional arrangement, but looks like it might be catching up.

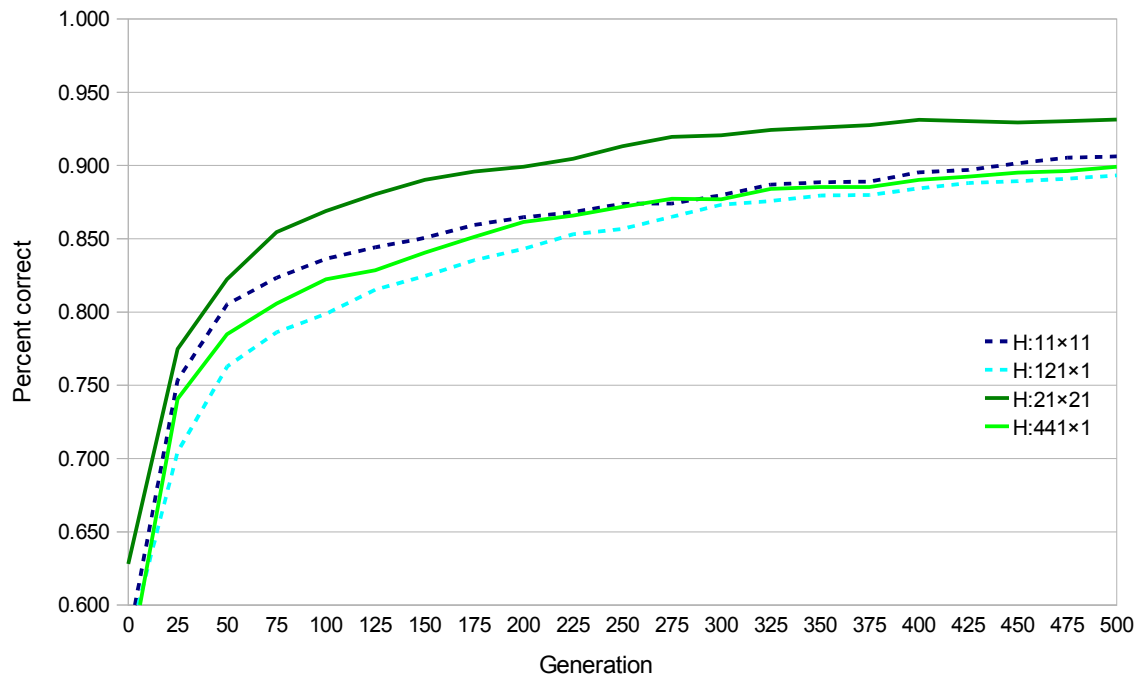


Figure 17: Comparison of performance for the one-dimensional versus two-dimensional spatial arrangement of neurons in the hidden layer for the rotation-invariant object recognition task, averaged over 30 runs. The hidden layer sizes of 11×11 (dark blue dashed) and equivalent size 121×1 (light blue dashed), and 21×21 (dark green solid) and equivalent size 441×1 (light green solid), have the same number of neurons respectively and the same connection topology (fully connected) and so have the same representational power, the only change is the spatial arrangement of neurons as perceived by the hypercube encoding scheme.

This indicates that HyperNEAT finds it easier to evolve solutions, for this task at least, when more spatial dimensions are present in the arrangement of neurons in a layer (at least for one versus two dimensions). This is not a surprising result given the bias towards evolving weight patterns aligned with the axes of the spatial dimensions of the layers, and in particular the bias towards generating a set of weight patterns for incoming connections that vary in some way from neuron to neighbouring neuron within a layer along each dimension of the layer. With more dimensions in the spatial arrangement of a layer it may be easier for HyperNEAT to create a greater variety of weight patterns for the neurons in the layer as the pattern for each neuron can vary over more dimensions.

5 Reinforcement learning tasks

This set of tasks required learning various navigation tasks for a simulated robot (in the shape of an upright cylinder), whose only input is a monocular monotone camera, in a 3D environment. The sub-tasks are referred to as task 2.x, where x is the number of the sub-task.

The robot used a differential steering drive, with a wheel on either side of the robot which is independently driven. The substrate networks had two outputs, one for each wheel, which are scaled to the range $[-0.5, 1]$, with a value of 1 or -0.5 indicating full speed forward or reverse respectively (the maximum reverse speed being half that of the forward speed). The maximum forward speed of the robot was 1 metre per second.

The substrate input was the raw camera image from the robot, with white corresponding to value 1 and black value 0.

The robot was a cylinder and the target a white sphere, both of radius 0.25 metres. The simulated environment was a 5×5 metre area with dark grey floor and ceiling and black boundary walls.

The parameters used are listed in Appendix 4 - Parameters for Reinforcement Learning tasks.

5.1.1 Technical difficulties

The original schedule of experiments for the robot-navigation tasks had to be scaled back due to technical difficulties. It was expected that experiments could be run remotely (as the author lives two hours away from UNSW) on a powerful server located at UNSW, and possibly on idle lab machines. However there were numerous technical difficulties with running the experiments remotely (due to combinations of Java3D refusing to run in a “headless” mode despite only rendering to off-screen buffers, apparent video card driver bugs, and incompatibilities with X11 forwarding and the Java virtual machine implementation on Mac OSX), which could not be solved by the author (along with UNSW System Support) in the time available.

In the end the author ended up buying a desktop computer with a moderately powerful 3D graphics card to run the experiments on (the author's only other computer was a lap-top with poor 3D hardware acceleration).

Due to the time spent trying to solve the above issues with running the experiments remotely and then only being able to run them on a single moderately powerful desktop computer there was not as much time and computational power available as was planned on. Thus it was not possible to perform as many experiments as planned.

5.2 Task 2.1

This task required the robot to locate and move towards the target, which is stationary, in an empty arena in a limited amount of time. The robot must turn until it sees the target and then move as quickly as possible towards it until running into it.

5.2.1 Method

The camera resolution was 25×12 pixels, with a 90° horizontal field of view (45° to the left and right each). The topology of the substrate was two fully connected layers of size 25×12 and 2×1 .

The fitness of an individual was determined with 8 trials. In the first two trials the initial orientation of the robot was facing 14° to the left and 14° to the right respectively of the direct line to the target. In the four subsequent trials these angles were increased by 14° , up to 42° degrees. For the last two trials the robot initially faced 90° to the right of the direct line to the target and then 180° degrees. Thus the robot had to locate the target from a variety of different starting positions, and for the last two the target is initially out of view.

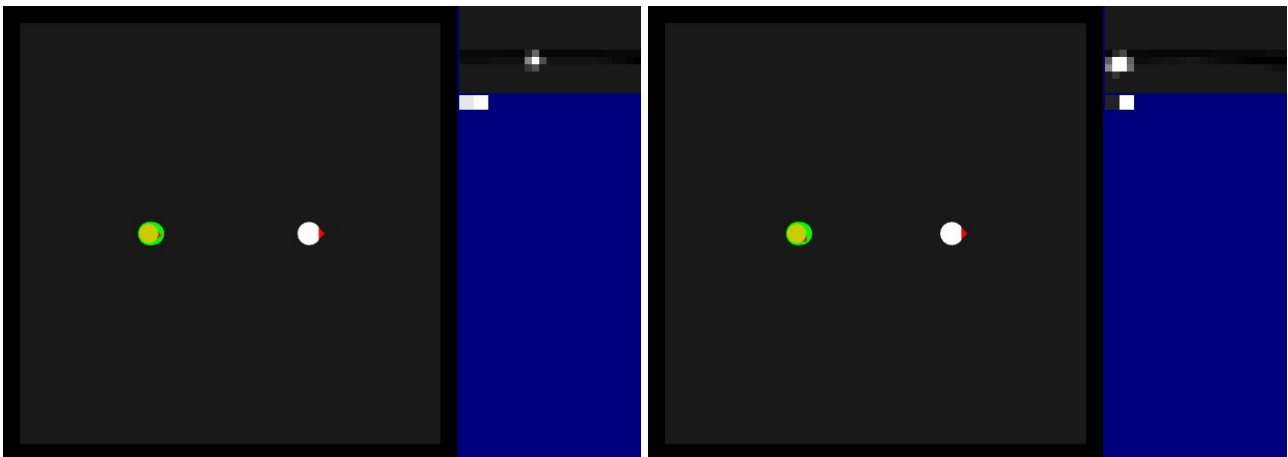


Figure 18: Images from robot navigation task 2.1. In each image the arena is depicted on the left (birds-eye view), showing the black boundary walls, white target and green/yellow robot (the red triangles on each indicate rotation). The sub-image at top-right is the input to the neural network, which is the robot-mounted camera image of resolution 25×12 . Below the camera image are two dots representing the output of the network. In the left and right images the robot is facing 14° and 42° respectively to the right of the direct line to the target at the beginning of a trial. In the right image it can be seen that the left output from the network (determining the speed of the left wheel) is dark, corresponding to the left wheel being reversed and so turning the robot towards the target.

The distance to the target in each trial was adjusted so that given the fixed time limit for a trial there

was just enough time to turn around and head approximately directly towards the target (rather than moving in, for example, a wide arc). For the last two trials the robot must turn clockwise until it finds the target.

The fitness function was based on the remaining distance to the target at the end of each trial for the second to last time step (the second to last is used as otherwise solutions that bring the robot extremely close but not quite touching the target are generated as actually touching the target gives no extra fitness). Specifically the fitness, f , of an individual was given by:

$$f = 1 - \sqrt{\frac{\sum_i (d_i / m)^2}{t}}$$

where d_i is the remaining distance to the target at the second to last time step for trial i , m is the maximum distance possible between the robot and the target given the arena size, and t is the total number of trials. This function is essentially a root-mean-squared-error function, basically averaging over the remaining distance but giving greater fitness to solutions that bring the robot closer to the target for most trials rather than very close on some trials and further away on others.

When a solution was found that was able to reach the target for every training trial evolution was stopped and the solution was tested 36 times: in the first test the robot is facing directly towards the target, and in each subsequent test the robot was successively rotated 5° (up to 180°) anti-clockwise. Additionally the distance between the robot and the target was successively increased from that used in training to double that used in training. The many different starting rotations (most of which do not align with the training examples) and further distances were designed to test the generality of the solution found.

5.2.2 Results and analysis

Various combinations and variations of training trials were tried initially, including incremental training regimes, before settling on the one described above. Most resulted in HyperNEAT becoming stuck in local optima, even with large population sizes (250 individuals) and many generations (1000). Unless the distances to the target were adjusted for each trial so that there was just enough time to turn and move towards the target, HyperNEAT would tend to get stuck in a local optimum by finding solutions that would reach the target for the easiest trials first and then not be able to find more general solutions that could reach the target for all trials. Making the fitness function penalise less general solutions more than it already did (for example taking the 3rd or 4th root of the mean of the 3rd or 4th power of the error) was not enough to avoid this problem.

As shown in figure 19 (percent correct plot) HyperNEAT nearly always evolved a solution that could reach the target for all training trials except one. For the last trial (facing 180° away from the target) it typically became stuck in a local optima and did not quite turn fast enough to be able to reach the target in time. The average remaining distance over all trials is very close to zero; although most evolved solutions did not quite make it to the target for the last trial, it came very close to it and if given more time (another 5% or 10% of the total time available) it would reach the target.

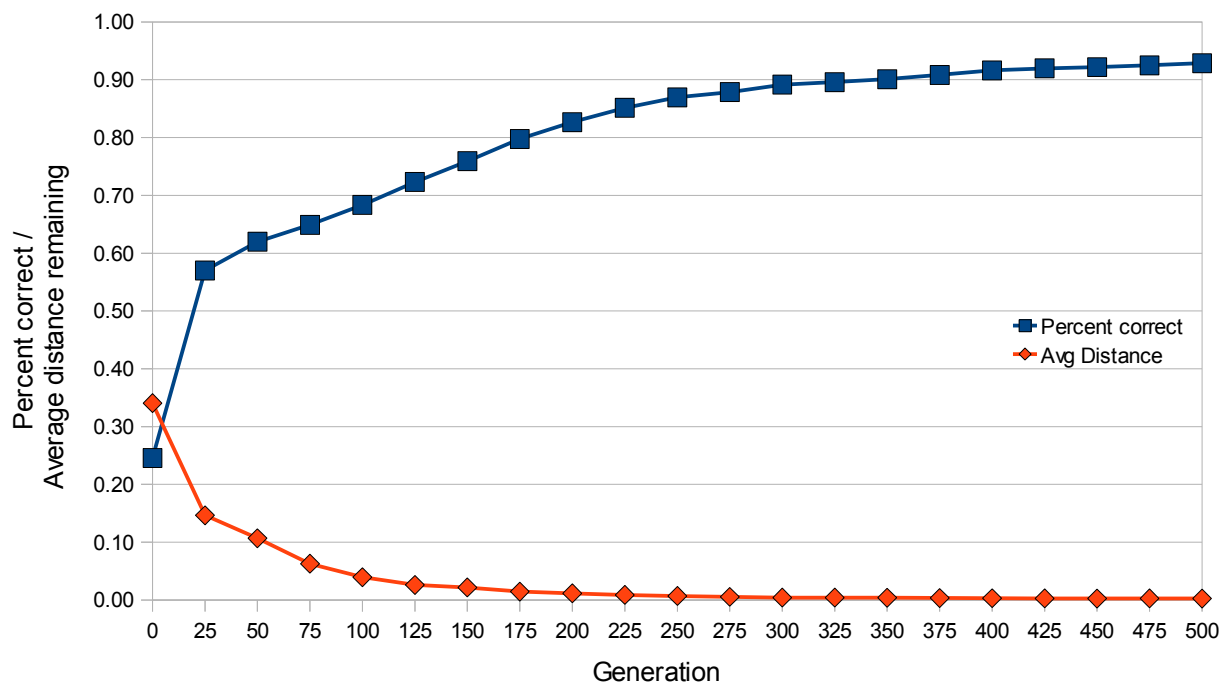


Figure 19: Performance results for robot navigation task 2.1, averaged over 30 runs. Shown are the percentage of trials correct (where the robot reached the target) and the average remaining distance to the target over all trials (using the same scale).

When testing these solutions about half were able to reliably reach the target from each starting orientation and all distances tested (up to twice the distance used during evolution). The other half of the solutions worked until the cases where the target was not initially in view and the robot was approximately over 1.5 times the distance to the target (in these cases the robot is also closer to the boundary walls as the robot and target were gradually moved to the left and right respectively for each successive test). Instead of continuing to turn until the target was visible the robot tended to wander off until it hit the boundary walls. Thus only a slight change in the input image (being slightly closer to the boundary walls while the target was not visible) was enough to throw these solutions off.

Figure 20 shows typical weight patterns for the incoming connections for the output neurons. The left, middle and right images are of increasingly less general solutions as determined by the tests. All solutions contain a section that fades smoothly from lighter to darker approximately horizontally in the region corresponding to where the target appears in the image (in opposite directions for the left and right outputs). This part is responsible for aiming the robot at the target when it is visible. Each solution has differing patterns for causing the robot to turn when the target is not visible, resulting in the different levels of performance during testing. In the least general solution the entire top region of the left output contains a very dark band at the top, in the second least general solution a similar but lighter band is present at the top (but still representing negative weight values), whereas the most general solution has a much smaller dark region at the top. The smoothness of weight value transitions and total level of contrast between weight values appears to be correlated to the generality of the solution. A similar result was found in [3] where HyperNEAT was applied to the task of checker board position evaluation; the solutions that had the smoothest weight patterns were also found to be the most general in post-evolution tests that differed from the training examples.



Figure 20: Typical weight patterns evolved for the incoming connections for the output neurons for robot navigation task 2.1. The left, middle and right images are of increasingly less general solutions respectively as determined by tests performed once all evaluation trials used during evolution were solved. All solutions contain a section at the bottom, in the region corresponding to where the target appears in the image, that fades smoothly from lighter to darker approximately horizontally (in opposite directions for the left and right outputs). Each solution has different patterns for causing the robot to turn when the target is not visible, resulting in the different levels of performance during testing. In the less general solutions the entire top region of the left output contains a darker band at the top, whereas the most general solution only has a smaller dark region at the top. The smoothness of weight value transitions and total level of contrast between weight values seems to be correlated to the generality of the solution.

5.3 Task 2.2

This task extended on the previous task by hiding the target behind a simple maze, requiring the robot to actively search for the target instead of turning on the spot until the target comes into view. As before, the target was stationary and the robot had a limited amount of time to reach the target.

5.3.1 Method

The camera resolution was increased to 40×20 pixels. The topology of the substrate was three fully connected layers layers of size 40×20 , 3×3 and 2×1 .

The maze was C-shaped and had walls of colour 50% grey, so could be distinguished by their shape and/or intensity.

The fitness of an individual was determined with two trials. In both, the robot starts on the opposite side of the maze to the target and facing towards it, but in the two trials the robot and target swap places, first with the robot “behind” the C and then partially inside it, as shown in figure 21. The fitness function was the same as that for task 2.1.

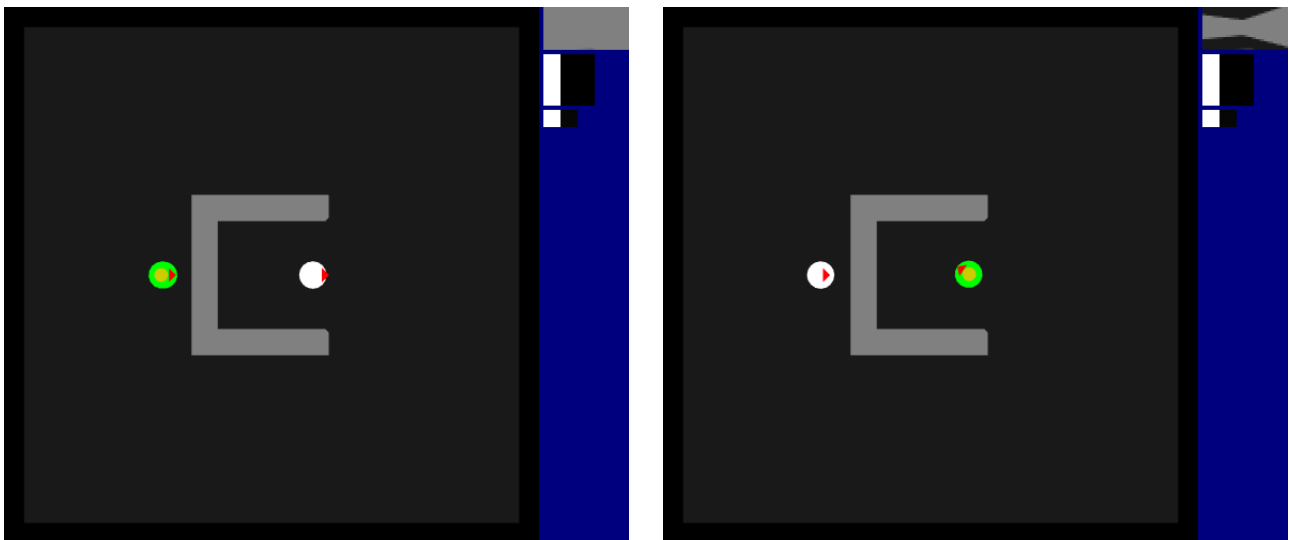


Figure 21: Images from robot navigation task 2.2. The sub-images at top-right now also show the activation levels of the hidden layer of size 3×3 . The left and right images show the initial starting positions of the robot and the location of the target in the two training trials. The grey maze walls can be seen in the foreground of the image input to the network.

5.3.2 Results and analysis

Similarly to the previous robot navigation task, it was found that HyperNEAT tended to get stuck in local optima. In this case experiments were initially conducted in which only the first trial was used and there was only just enough time for the robot to reach the target via the optimal route (in the experiment as described in the method above the robot has enough time without taking the optimal route). In these initial experiments HyperNEAT typically (about 75 percent of the time) found a solution that first turned the robot on the spot before navigating around the maze, but not quite making it to the target as it would run out of time due to having turned on the spot at the beginning.

The general solution required following the walls of the maze by keeping the maze wall in part of the field of view (to the left or right), and turning when a wall was not in view until it was in view. The optimal route consisted of moving approximately directly towards one of the corners of the maze and not spinning on the spot first. The non-optimal solution arose because it was easier to evolve weight patterns that implemented the wall-following behaviour but also caused the robot to turn in the same direction as it would to keep the wall partially in view, instead of turning in the opposite direction, when a wall was straight ahead and occupied most of the field of view.

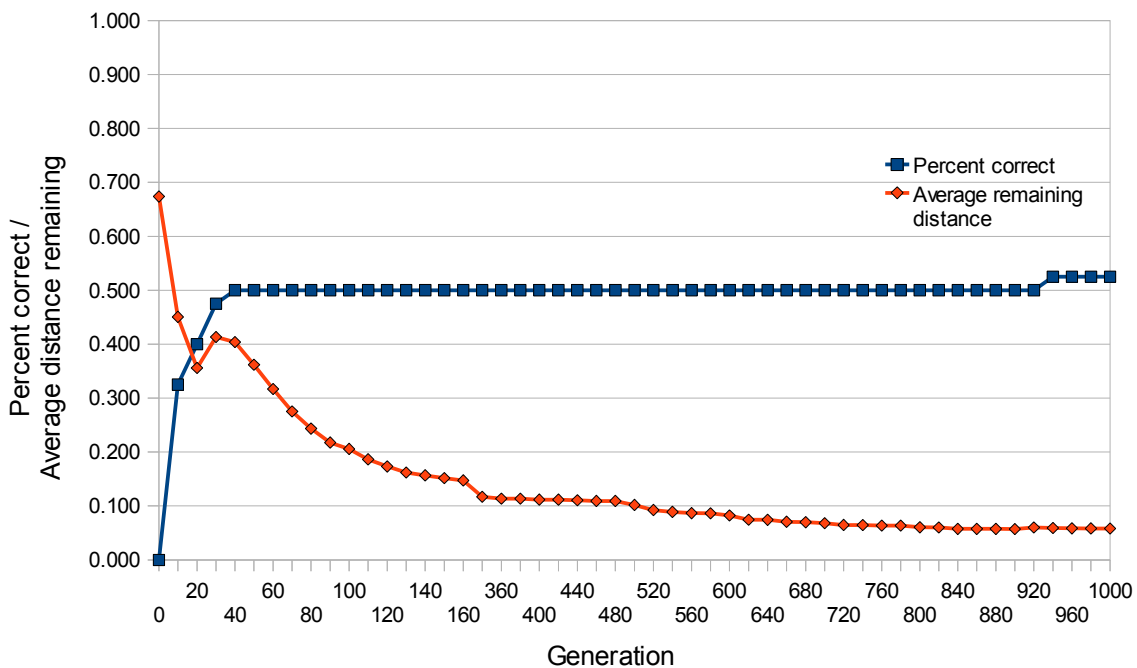


Figure 22: Performance results for robot navigation task 2.2, averaged over 20 runs. Shown are the percentage of trials correct (where the robot reached the target) and the average remaining distance to the target over both trials (using the same scale).

Figure 22 shows the performance results for the full experiment (using the two trials as described in the method). It can be seen that HN typically evolved solutions that could reach the target for one trial but not both, however for one of the runs (out of 20) it found a solution that could reach the target in both trials. It always evolved solutions that brought the robot very close to the target, and hence implemented a wall-following strategy.

In the full experiment enough time was provided to allow turning on the spot before starting to navigate around the maze. However HyperNEAT again became stuck in a local optima. In this case, once a wall-following strategy had evolved, HyperNEAT was typically unable to modify these solutions to turn towards the target enough once it came into view in order to collide with it for both trials. Instead they would typically turn slightly towards the target but still only begin to move past it (before running out of time), similarly to the wall following strategy. In one solution the robot does turn directly for the target once it comes into view for both trials, so it appears that the task was solvable with the substrate network topology used. For some solutions it looks like if given a little bit more time the robot would reach the target by following a short “decaying orbit” trajectory. It may be the case that simply increasing the mutation rates for adding structural elements or changing weight values in the CPPNs would have enabled HyperNEAT to escape the local optima. Figure 23 shows the path of a typical solution.

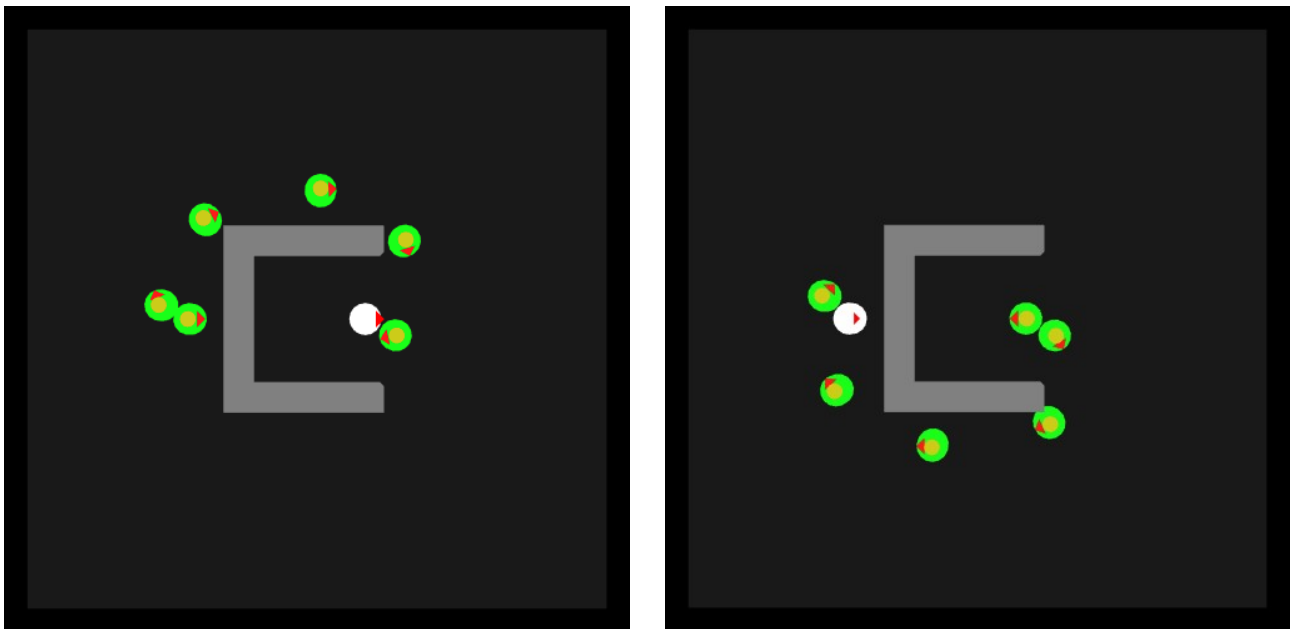


Figure 23: Time lapse view of path taken by the robot for a typical solution to robot navigation task 2.2. In this solution the robot initially spins clockwise, and then keeps the wall to its right. The wall-following strategy evolves first, but then HyperNEAT has trouble modifying the solutions to turn the robot towards the target enough to collide with it, and instead they follow a “decaying orbit” path.

6 Discussion and Future Work

This study has highlighted some important properties, possible limitations and potential avenues for improvement of HyperNEAT that could be studied in more detail. These findings are summarised as follows.

Optimising weights for multiple hidden layers

In the rotation-invariant object recognition task it was found that HyperNEAT appeared to struggle to optimise the weight layers for multiple hidden layers. Further study is needed to confirm this, explore the situations in which this problem occurs and ideally find ways to improve HyperNEAT to solve it.

Bias towards weight patterns aligned with axes of layers

In the object recognition tasks it was also found that HyperNEAT exhibited a bias towards evolving weight patterns, for the incoming connections to a neuron, aligned with the axes of the two dimensions of the planar layers of neurons.

Related to the above, for the scale and rotation-invariance tasks, it was also found that the weight patterns for incoming connections over multiple neurons in the hidden layer varied over the two dimensions of the layer. As one moved along the x axis from neuron to neighbouring neuron, the incoming connection weight patterns for each neuron varied in one way, and as one moved along the y axis the patterns varied in another way. Thus HyperNEAT appears to be biased towards producing variations in the incoming connection weight patterns for neurons across a layer aligned with the axes of the layers. This is to be expected given the bias noted above towards producing patterns for the incoming connection weights for a single neuron aligned with the axes of the layers, as they are opposite sides of the same coin: if the outgoing connections from a neuron in the input layer were visualised, the variations in input connection weight patterns over the rows and columns of neurons in the hidden layer would show as variations in the single pattern of outgoing connections for an input neuron.

This bias is not necessarily a problem; HyperNEAT was not limited to producing weight patterns for incoming or outgoing connections that are aligned with the axes of the layers (as demonstrated in various other weight patterns shown in this study). However it is worth being aware of so that it can either be exploited or measures can be taken to avoid or mitigate it. Further study could be undertaken to determine the properties of the bias and methods of using or mitigating it.

Topological organisation of neurons within a layer

The bias towards generating weight patterns aligned with the axes of the layers (over incoming and outgoing connections for a neuron) suggests a way to make it easier for HyperNEAT to evolve solutions that can encode or represent within the activation space of a layer more aspects of some information. Instead of layers consisting of either a one-dimensional line, or two dimensional plane, of neurons it would be possible for layers to be spatially organised as either a three-dimensional volume or an n-dimensional hyper-volume of neurons within a layer. With more dimensions in the spatial arrangement of a layer, it may then be easier for HyperNEAT to create a greater variety of weight patterns for the neurons in the layer, as the pattern for each neuron can vary over more dimensions.

Supporting this hypothesis, the study found that for the rotation-invariant object recognition task, the performance of HyperNEAT decreased when the number of dimensions used in the spatial arrangement of neurons in the hidden layer was decreased from two to one.

To increase the number of dimensions in the spatial arrangement of neurons in a layer the weight function (specifically CPPN in HyperNEAT) describing weight values would have inputs to specify the positions of the source and target neuron in these extra dimensions (instead of just for the x and y axes of the planar organisation). Note that for a feed-forward network there would be no connections between neurons within the volume, and that if a layer (whether organised in a line, plane or n-dimensional volume) contained the same number of neurons with the same connection topology between layers (e.g. fully-connected between layers), then the representational power of the overall network would be unaffected; it would only be changing the spatial organisation of the neurons as perceived by the hypercube encoding scheme used by HyperNEAT. This would increase the size of the search space (by increasing the number of inputs to the weight function), but for some kinds of tasks the trade-off may be worthwhile. Further study is needed to determine if this offers a viable avenue for improving the performance and applicability to various kinds of tasks and input spaces of HyperNEAT.

Another possible avenue of investigation is to use the extra dimensions in an n-dimensional layer to represent other information in the input (or output) space. For example in this study all images were represented as grey-scale images, which naturally map to the two-dimensional planar organisation of neurons within a layer used in this and many other studies on HyperNEAT. If we wish to include, for example, colour information (e.g. the hue as a scalar value for each pixel) in the images then there is no “natural” way to map this together with the intensity values onto a two-dimensional layer of input neurons. In previous studies this has been handled by introducing multiple two-dimensional

input layers that all feed into a single two-dimensional input layer, and having an extra output from the CPPN to describe the weight values for those connections. Further study is needed to determine if the alternative suggested can work as well or better than the current approach.

New comparison method for speciation when using hypercube encoding

It was found in both robot-navigation tasks that HyperNEAT had a strong tendency to become stuck in local optima. This is a common problem for evolutionary algorithms. The NEAT component of HyperNEAT (the part that generates the functions that encode the weight pattern for the substrate) uses a fitness-sharing type of speciation to help avoid becoming stuck in local optima [40], however this did not seem to be enough to avoid this situation for many of the combinations and variations of training trials.

NEAT determines whether an individual belongs in (or is compatible with) an existing species using a unique method that essentially compares the similarity in the (remaining) genetic heritages of the individual and that of a representative of the species. In this way genetic diversity is explicitly maintained. While it has been shown that speciation using this method of comparison improved the performance of the evolutionary process it is clearly not perfect.

When an evolutionary algorithm is being used to evolve weight functions for the hypercube encoding scheme, it opens up another possible way of comparing the similarity of genotypes, to either replace or complement that used in NEAT. Two genotypes can be compared by directly comparing the output of their weight functions over different input values, or more specifically querying the weight functions at a low resolution within the range of values used in the hypercube encoding (typically the unit hyper-cube). A scalar “similarity” value could be generated using, for example, a root-mean-squared-difference function or some other function customised to this task. The motivation for such a comparison method is that the actual weight values produced by the weight function may be a better indication of behaviour than the specific genetic contents of individuals, and hence provide a better way of maintaining behavioural diversity to avoid becoming stuck in local optima. The resolution at which the weight functions are queried will determine the quality of the comparison, providing a scalable method of quality of comparison versus time used to perform the comparisons.

7 Conclusion

This study explored the properties and abilities of the HyperNEAT neuroevolution method by applying it to visual processing tasks from two different domains over multiple tasks with varying degrees of complexity. The aspects of the method studied were the performance of the method, the effect of different substrate topologies, the effect of variations in the tasks and the characteristics of the solutions produced.

The task domains studied were: object recognition using raw image data in a supervised learning context, with variations on object complexity, object localisation in the image and rotation and scale-invariant object recognition; and robot-navigation in a reinforcement learning context, requiring learning simple target acquisition and obstacle avoidance strategies using only the raw image from a robot-mounted camera.

Positive results were obtained for non-trivial tasks to which neuroevolution methods have not been successfully applied previously, specifically requiring the direct processing of the high-dimensional input space of raw image data (as opposed to an abstract image of the environment or information from pre-processing the images). This indicates that HyperNEAT, or more generally methods using hypercube-based encodings, may be a powerful new method for tasks from these kinds of domains.

Several important characteristics of HyperNEAT that have not been reported previously were made evident in this study: a bias towards evolving weight patterns aligned with the axes of the spatial arrangement of neurons; the impact on performance of the number of dimensions in the spatial arrangement of neurons in hidden layers; the ability to co-evolve weight patterns for two layers of connections; the possible difficulty with co-evolving weight patterns for more than two layers of connections; the ability to evolve solutions more quickly using larger substrate networks; and the tendency to become stuck in local optima for certain kinds of tasks or training example sets.

Various avenues of further research and possible extensions or modifications to HyperNEAT based on these findings were outlined, including:

- spatially arranging substrate network layers in (hyper-)volumes instead of lines or planes, to help mitigate or exploit the bias towards evolving weight patterns aligned with the axes of the layers, and/or to enhance or enable the ability to encode multi-dimensional information in the activation space of a layer (either input, hidden or output); and
- a new method of comparing individuals for the purpose of speciation in the evolutionary process based directly on the weight patterns produced by the weight functions.

8 Appendices

8.1 Appendix 1 - Parameters for Object Recognition tasks 1 and 2

NEAT (CPPN) parameters

population size	100
probability of adding a neuron	0.03
probability of adding a connection	0.4
probability of removing a connection	0.05
probability of each weight being modified	0.6
weight mutation magnitude standard deviation	2
survival rate	0.3
proportion of reproduction from crossover versus asexual	0.5
minimum individuals to select as elite from each species	1
minimum species size to select elites for	5
maximum stagnant generations before removing species	N/A
disjoint genes compatibility factor	2
excess genes compatibility factor	2
weight delta compatibility factor	1
initial compatibility threshold	1.9
target number of species	8
maximum weight range for CPPN	[-3, 3]
functions available for CPPN	linear, clamped linear, absolute, convert signed ⁴ , Sigmoid, Gaussian, sine
recurrent connections	disallowed

Hypercube parameters

include deltas in CPPN inputs	yes
include angle in CPPN inputs	no
bias for substrate neurons	no
maximum weight magnitude for substrate	[-3, 3]
connection expression threshold	0.2
layer connection topology	fully-connected
activation function	Sigmoid

⁴ The “convert signed” function converts the input from the range [0, 1] to [-1, 1], effectively giving it a sign value.

The input is clamped to between 0 and 1, multiplied by 2, then 1 is subtracted.

8.2 Appendix 2 - Parameters for Object Recognition task 3

NEAT (CPPN) parameters

population size	250
probability of adding a neuron	0.25
probability of adding a connection	0.5
probability of removing a connection	0.02
probability of each weight being modified	0.1
weight mutation magnitude standard deviation	1.0
survival rate	0.3
proportion of reproduction from crossover versus asexual	0.5
minimum individuals to select as elite from each species	1
minimum species size to select elites for	5
maximum stagnant generations before removing species	N/A
disjoint genes compatibility factor	2
excess genes compatibility factor	2
weight delta compatibility factor	1
initial compatibility threshold	1.9
target number of species	15
maximum weight range for CPPN	[-3, 3]
functions available for CPPN	linear, clamped linear, absolute, step, convert signed, inverse absolute, divide ⁵ , Sigmoid, Gaussian, sine
recurrent connections	disallowed

Hypercube parameters

include deltas in CPPN inputs	yes
include angle in CPPN inputs	yes
bias for substrate neurons ⁶	yes
maximum weight magnitude for substrate	[-1, 1]
connection expression threshold	0.2
layer connection topology	connection range limited
activation function	Sigmoid

⁵ The divide function divides the value from the first incoming connection by the value from the second connection, ignoring all other connections.

⁶ A separate output from the CPPN is used to specify a bias value for each neuron in the substrate.

8.3 Appendix 3 - Parameters for Object Recognition task 4

NEAT (CPPN) parameters

population size	250
probability of adding a neuron	0.25
probability of adding a connection	0.5
probability of removing a connection	0.02
probability of each weight being modified	0.1
weight mutation magnitude standard deviation	1.0
survival rate	0.3
proportion of reproduction from crossover versus asexual	0.5
minimum individuals to select as elite from each species	1
minimum species size to select elites for	5
maximum stagnant generations before removing species	N/A
disjoint genes compatibility factor	2
excess genes compatibility factor	2
weight delta compatibility factor	1
initial compatibility threshold	1.9
target number of species	15
maximum weight range for CPPN	[-3, 3]
functions available for CPPN	linear, clamped linear, absolute, step, convert signed, inverse absolute, divide, Sigmoid, Gaussian, sine
recurrent connections	disallowed

Hypercube parameters

include deltas in CPPN inputs	yes
include angle in CPPN inputs	yes
bias for substrate neurons ⁷	yes
use separate output from CPPN for each layer of weights ⁸	yes
maximum weight magnitude for substrate	[-2, 2]
connection expression threshold	0.2
layer connection topology	fully-connected
activation function	Sigmoid

⁷ A separate output from the CPPN is used to specify a bias value for each neuron in the substrate.

⁸ The other option is to use an input to the CPPN whose value specifies the layer, and a single CPPN output for all weight layers.

8.4 Appendix 4 - Parameters for Reinforcement Learning tasks

NEAT (CPPN) parameters

population size	500
probability of adding a neuron	0.1
probability of adding a connection	0.2
probability of removing a connection	0.02
probability of each weight being modified	0.2, 0.5 for 2.1, 2.2 respectively
weight mutation magnitude standard deviation	1.0
survival rate	0.3
proportion of reproduction from crossover versus asexual	0.25
minimum individuals to select as elite from each species	1
minimum species size to select elites for	5
maximum stagnant generations before removing species	N/A
disjoint genes compatibility factor	1
excess genes compatibility factor	1
weight delta compatibility factor	1
initial compatibility threshold	1.9
target number of species	40
maximum weight range for CPPN	[-3, 3]
functions available for CPPN	linear, clamped linear, absolute, step, convert signed, inverse absolute, divide, Sigmoid, Gaussian, sine
recurrent connections	disallowed

Hypercube parameters

include deltas in CPPN inputs	yes
include angle in CPPN inputs	yes
bias for substrate neurons	yes
use separate output from CPPN for each layer of weights	yes
maximum weight magnitude for substrate	[-3, 3]
connection expression threshold	0.2
layer connection topology	fully-connected
activation function	Sigmoid

9 Bibliography

- [1] J. Gauci and K. Stanley, “Generating large-scale neural networks through discovering geometric regularities,” *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, p. 1004.
- [2] P. Verbancsics and K.O. Stanley, “Transfer Learning through Indirect Encoding,” *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, New York, NY: ACM, 2010.
- [3] J. Gauci and K. Stanley, “Autonomous Evolution of Topographic Regularities in Artificial Neural Networks,” *Neural Computation*, 2010.
- [4] D.B. D'Ambrosio and K.O. Stanley, “Generative encoding for multiagent learning,” *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008, pp. 819–826.
- [5] J. Clune, B.E. Beckmann, C. Ofria, and R.T. Pennock, “Evolving coordinated quadruped gaits with the HyperNEAT generative encoding,” *Proceedings of the IEEE Congress on Evolutionary Computing Special Section on Evolutionary Robotics*, 2009.
- [6] J. Drchal, J. Koutnik, and M. Šnorek, “Hyperneat controlled robots learn to drive on roads in simulated environment,” *Submitted to IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009.
- [7] D.B. D'Ambrosio and K.O. Stanley, “A novel generative encoding for exploiting neural network sensor and output geometry,” *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, p. 981.
- [8] N.K. Kasabov, S.I. Israel, and B.J. Woodford, “Adaptive, evolving, hybrid connectionist systems for image pattern recognition,” *Soft computing for image processing*, Physica-Verlag, Heidelberg, 2000.
- [9] K.O. Stanley and R. Miikkulainen, “A taxonomy for artificial embryogeny,” *Artificial Life*, vol. 9, 2003, pp. 93–130.
- [10] J.C. Bongard and R. Pfeifer, “Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny,” *Proceedings of the Genetic and Evolutionary Computation Conference*, 2001, pp. 829–836.

- [11] P. Bentley and S. Kumar, "Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem," *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 35–43.
- [12] F. Gruau, "Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm.," Doctoral dissertation, Ecole Normale Supérieure de Lyon, 1994 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.5939>>.
- [13] N. Jakobi, "Harnessing morphogenesis," *In To Appear in Proceedings of International Conference on Information Processing in Cells and Tissues*, 1995.
- [14] A. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society B*, vol. 237, 1952, pp. 37-72.
- [15] Angelo Cangelosi, Domenico Parisi, and Stefano Nolfi, "Cell division and migration in a 'genotype' for neural networks," 1994 <http://informahealthcare.com/doi/abs/10.1088/0954-898X_5_4_005>.
- [16] D. Roggen and D. Federici, "Multi-Cellular Development: Is There Scalability and Robustness to Gain?," *PROCEEDINGS OF PARALLEL PROBLEM SOLVING FROM NATURE 8, PARALLEL PROBLEM SOLVING FROM NATURE (PPSN) 2004*, 2004, pp. 391--400 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.4297>>.
- [17] J. Thangavelautham and G.M.T. D'eleuterio, "A Coarse-Coding Framework for a Gene-Regulatory-Based Artificial Neural Tissue," *IN ADVANCES IN ARTIFICIAL LIFE: PROCEEDINGS OF THE 8TH EUROPEAN CONFERENCE ON*, 2005, pp. 67--77 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.4961>>.
- [18] E.J.W. Boers and H. Kuiper, "Biological Metaphors and the Design of Modular Artificial Neural Networks," 1992 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.5056>>.
- [19] G.S. Hornby and J.B. Pollack, "Creating high-level components with a generative representation for body-brain evolution," *Artificial Life*, vol. 8, 2002, pp. 223–246.
- [20] G.S. Hornby and J.B. Pollack, "The Advantages of Generative Grammatical Encodings for Physical Design," *IN CONGRESS ON EVOLUTIONARY COMPUTATION*, 2001, pp. 600--607 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.9803>>.
- [21] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, 1990, pp. 461–476.
- [22] K. Sims, "Evolving 3D morphology and behavior by competition," *Artificial Life*, vol. 1,

1994, pp. 353–372.

- [23] M.H. Luerssen and D.M.W. Powers, “On the Artificial Evolution of Neural Graph Grammars,” 2003 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.3859>>.
- [24] F. Gruau, “Genetic synthesis of modular neural networks,” *Proceedings of the 5th International Conference on Genetic Algorithms*, 1993, p. 325.
- [25] S. Luke and L. Spector, “Evolving graphs and networks with edge encoding: Preliminary report,” *Late Breaking Papers at the Genetic Programming 1996 Conference*, 1996, pp. 117–124.
- [26] M. Komosinski and A. Rotaru-Varga, “Comparison of different genotype encodings for simulated three-dimensional agents,” *Artificial Life*, vol. 7, 2001, pp. 395–418.
- [27] J. Meyer, S. Doncieux, D. Filliat, A. Guillot, and A. Lip, “Evolutionary approaches to neural control of rolling, walking, swimming and flying animats or robots,” *IN: BIOLOGICALLY INSPIRED ROBOT BEHAVIOR ENGINEERING*, 2003, pp. 1--43 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.146.5677>>.
- [28] Y. Kassahun, M. Edgington, J.H. Metzen, G. Sommer, and F. Kirchner, “A General Framework for Encoding and Evolving Neural Networks,” *GECCO'07*, 2007 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.70.8729>>.
- [29] S. Nolfi and D. Parisi, “Growing neural networks,” *The Handbook of Brain Theory and Neural Networks*, 1991.
- [30] A. Cangelosi, D. Parisi, and S. Nolfi, “Cell division and migration in a 'genotype' for neural networks,” *Network: computation in neural systems*, vol. 5, 1994, pp. 497–515.
- [31] F. Dellaert, “Toward a biologically defensible model of development,” Citeseer, 1995.
- [32] F. Dellaert and R.D. Beer, “Co-evolving body and brain in autonomous agents using a developmental model,” *Cleveland, OH*, vol. 44106, 1994.
- [33] F. Dellaert and R.D. Beer, “Toward an evolvable model of development for autonomous agent synthesis,” *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, 1994, pp. 246–257.
- [34] F. Dellaert and R.D. Beer, “A developmental model for the evolution of complete autonomous agents,” *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 1996.

- [35] S.A. Kauffman, *The origins of order: Self organization and selection in evolution*, Oxford University Press, USA, 1993.
- [36] J.C. Bongard and C. Paul, "Investigating morphological symmetry and locomotive efficiency using virtual embodied evolution," *From Animals to Animats: The Sixth International Conference on the Simulation of Adaptive Behaviour*, 2000.
- [37] J.C. Bongard, "Evolving modular genetic regulatory networks," *Proceedings of the 2002 congress on evolutionary computation*, 2002, pp. 17–21.
- [38] J.C. Astor and C. Adami, "A Developmental Model for the Evolution of Artificial Neural Networks," *Artificial Life*, vol. 6, Jul. 2000, pp. 189-218
<<http://dx.doi.org/10.1162/106454600568834>>.
- [39] D. Federici, "Evolving a Neurocontroller Through a Process of Embryogeny," 2004
<<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.5896>>.
- [40] K.O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, 2002, pp. 99-127
<<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.4591>>.
- [41] K.O. Stanley, "Exploiting regularity without development," *Proceedings of the AAI Fall Symposium on Developmental Systems*, 2006.
- [42] K.O. Stanley, B.D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 9, 2005, pp. 653--668 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.6243>>.
- [43] K.O. Stanley and R. Miikkulainen, "Competitive Coevolution through Evolutionary Complexification," *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, vol. 21, 2002, pp. 63--100 <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.6487>>.
- [44] J. Gauci and K.O. Stanley, "A case study on the critical role of geometric regularity in machine learning," *Proc. of the 23rd AAI Conference on Artificial Intelligence (AAAI-2008)*. AAI Press, 2008.
- [45] K. Chellapilla and D.B. Fogel, "Evolving an expert checkers playing program without using human expertise," *IEEE Transactions on Evolutionary Computation*, vol. 5, 2001, pp. 422–428.
- [46] B. Woolley and K. Stanley, "Evolving a Single Scalable Controller for an Octopus Arm with a Variable Number of Segments," *Parallel Problem Solving from Nature–PPSN XI*, 2010, pp.

270–279.

- [47] D.B. D’Ambrosio, J. Lehman, S. Risi, and K.O. Stanley, “Evolving Policy Geometry for Scalable Multiagent Learning,” 2010.
- [48] W. Jaśkowski, K. Krawiec, and B. Wieloch, “NeuroHunter-An Entry for the Balanced Diet Contest,” 2008.
- [49] J. Clune, C. Ofria, and R.T. Pennock, “The sensitivity of HyperNEAT to different geometric representations of a problem,” *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 675–682.
- [50] Z. Buk, J. Koutník, and M. Šnorek, “NEAT in HyperNEAT substituted with genetic programming,” *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009)*, 2009.
- [51] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber, “Super Mario Evolution,” *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, 2009.
- [52] S. Risi, J. Lehman, and K.O. Stanley, “Evolving the Placement and Density of Neurons in the HyperNEAT Substrate,” *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, New York, NY: ACM, 2010.
- [53] S. Risi and K. Stanley, “Indirectly Encoding Neural Plasticity as a Pattern of Local Rules,” *From Animals to Animats II*, 2010, pp. 533–543.
- [54] F. Gruau, D. Whitley, and L. Pyeatt, “A comparison between cellular encoding and direct encoding for genetic neural networks,” *Proceedings of the First Annual Conference on Genetic Programming*, 1996, pp. 81–89.
- [55] J. Clune, C. Ofria, and R.T. Pennock, “How a generative encoding fares as problem-regularity decreases,” *PPSN (G. Rudolph, T. Jansen, SM Lucas, C. Poloni, and N. Beume, eds.)*, vol. 5199, 2008, pp. 358–367.
- [56] *Simbad 3d Robot Simulator*. <<http://simbad.sourceforge.net/>>.
- [57] D. James and P. Tucker, *Another NEAT Java Implementation*, 2004 <http://anji.sourceforge.net>.