Andreas Røyset Stensbye

Evolving Multi-Spatial-Substrates
to extend HyperNEAT in order to increase
Functional Modularity

Master project, 02.12.2019

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

Humanity have begun to actively use artificial intelligence to solve problems. However, many of these solutions are time consuming to develop and often fine-tuned to specific problems. With the world in constant change, we face many new challenges that require faster solutions. This work presents Evolved-Multi-Spatial Substrate HyperNEAT (EMSS-HyperNEAT) for evolving both topology and the developmental stage of an artificial neural network(ANN). It is based on the HyperNEAT algorithm, which traditionally expects a hand-crafted neuron geometry (substrate structure). Such hand-crafted substrate structures often require experience and deep understanding of the problem, and this may be hard to acquire for more complex problems. EMSS-HyperNEAT aims to automate this process, saving time and through evolution finds topologies that can reflect spatial aspects of the problem. The goal is that these discovered spatial properties can be utilized for more functional specialisation within the ANN, and thus achieve functional modularity. We often observe functional modularity in biological neural networks such as the brain, where different brain functions can be handled separately by modules consisting of neurons located physically close together. Despite nature often choosing modular neural networks, evolutionary algorithms have struggled with creating effective modular solutions for artificial neural network. This work investigates the effects of allowing evolution to search for both the substrate structure and the indirect encoding for the ANN connectivity, and explore the effect this has for functional modularity.

Preface

The following master thesis is were conducted at the Norwegian University of Science and Technology(NTNU) in Trondheim, Norway during the period 06.01.2019-02.12.2019.

I would like to thank my supervisor Pauline Catriona Haddow for being so kind to take me on as a master student. I am deeply grateful for her help, guidance and critique throughout the entire project, and this project would not have been the same without her.

Much gratitude also goes to my fiancée Alyona for keeping my spirits up and encouraging me to work hard.

Andreas Røyset Stensbye
Trondheim, December 2, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work presents a novel algorithm for evolving Multi-Spatial Substrates used in the evolutionary algorithm HyperNEAT. HyperNEAT is a bio-inspired algorithm for indirectly encoding weights in an artificial neural network(ANN). It is perhaps most famous for its ability to exploit spatial information in problems by reflecting them in the placement of neurons in the *substrate*. Since HyperNEAT was introduced multiple improvements have been proposed, but placing the substrate under direct evolution have not received much attention. This work places the substrate directly under evolutionary control, and analyses the results this has on the phenotype networks' modularity. Modularity is the property of a system whereby it can be broken down into a number of relatively independent, replicable, and composable subsystems, known as *modules*[1]. These subsystems can refer to both the how modules are structurally connected together, and to how the system perform tasks separately in independent subsystems. For ANN's this means modularity can be defined as how the neurons are connected together (topology), and how the different parts of the ANN perform separate functions to solve a problem. These two types of modularity are called *topological modularity* and *functional modularity*, and will be referred to as *T-modularity*, and *F-modularity*. F-modularity is the more descriptive of how an ANN processes information and solve problems, and are therefore given greater focus in the thesis. The following chapter will present the motivation and background in section 1.1, followed by the goals and research questions in section 1.2. Then section 1.3 elaborate on the literature review leading up to this project, and at the end of the chapter section 1.5 will present the structure of the thesis.

1.1 Motivation and Background

In biological life the development of organisms are guided by chemical gradients[11; 21; 9]. This assist the cells in determining their function they should perform in the organism. The cell's physical location can for example determines if the cell should become part of an arm or a kidney. This directs how organisms creates modular subsystems such as organs and limbs. Brains are also divided up in modular systems where a group of physically close cells makes up a brain region, for example the frontal lobe, motor lobe or the occipital lobe. The hope is to take inspiration from these mechanisms to make artificial neural networks more modular. Modularity allow new mutations to be confined to a module, and thus a module can change and improve without the change being destructive in another part of the system. This makes modular systems better at improving by evolutionary processes. Modularity can also allow for reuse of sub systems, easier learning, more robustness, and more "human understandable" systems.

The first attempt at using neural networks to create an abstraction of natural development was Compositional Pattern Producing Networks (CPPN)[32]. CPPN are designed to create patterns analogous to the local interactions between gradient patterns over time in embryonic development. CPPN are in essence a normal feed forward neural network, but with activation functions that focuses patterns such as sin, abs, Gaussian, tan etc. This pattern producing network was later applied to determining the weights for another network. This algorithm was called HyperNEAT[33].

The purpose of HyperNEAT is to utilize spatial information to create a neural network. HyperNEAT focuses on the evolving the connectivity between neurons, in a way such that evolution can exploit the spatial placement of neurons and their geometrical correlation. This is similar to how the brain often are organized in a way that reflects the physical world. There is a left right symmetry to match the body, and dual sensory input (left and right eyes, ears), as well as the layout of neurons to mirror sensory configuration, as seen in the visual cortex[5]. In HyperNEAT the spatial information is represented by placing all neurons in a substrate and giving the neurons coordinates so that the neurons exist in a "geometrical space". The geometry of the substrate itself is however not included in evolution and it is instead fixed a priori. The substrate structure were meant to be hand-crafted to fit the problem. But this is a lot of extra work for the user, and it is not always obvious how the substrates should be designed. This require a lot of domain knowledge and in many problems this is a non-trivial decision. For example what is the geometric shape best suited for recognizing cats in images?

One large difference between the ANN produced by HyperNEAT and biological brains, is modularity. HyperNEAT has been shown to struggle with producing modular ANNs[6]. We still don't have a complete understanding of how modularity forms in evolutionary system, but one strong factor seems to be physical constraints. Most variations of HyperNEAT that aims at encouraging modularity actively uses the neuron's geometrical relationship to simulate physical constraints. These variations give the substrates even more impact on the ANN developed, yet search for good substrate structures have received little attention in the literature. The hypothesis at the basis of this work is that it would be beneficial to include substrate structure in the evolutionary process, and in particular in regards to techniques that very actively utilize the substrates to encourage modular neural networks.

1.2 Goals and Research Questions

This section includes the goal that have guided the work in this project, and research questions that will be investigated in this thesis.

Goal *Investigate how evolution of the substrate can extend HyperNEAT in order to achieve modular neural networks.*

HyperNEAT places a big burden on the researcher to hand craft a good *substrate* design for the organisation of neurons before HyperNEAT can be used, and only the connectivity pattern changes during evolution. Even though the substrates plays an active role in the shape of the ANN being produced by HyperNEAT, the substrates are traditionally fixed at the start of evolution. In this work the substrate geometries are not excluded from the evolutionary process and are allowed to change and evolve. The hypothesis is that by allowing evolution greater freedom, the HyperNEAT will be better able to achieve modular solutions. It would also remove the burden of having to design substrates.

Research question 1 *How should the CPPN be adapted to achieve modular neural networks?*

Research question 1.2 *How should the CPPN outputs be adapted to achieve the highest performance on image classification accuracy?*

There exist different methods on how the CPPN can be used with substrates, and it is unclear what solution will work best when the substrate structure is evolving. Especially in regard to which substrates utilize which CPPN output. The chosen method should most importantly be able to handle modular problems

to show that it can achieve functional modularity, but the method should also generalize to well to work well for tasks other than just modularity.

Research question 2 *How does ANNs produced perform on problems with geometrical relationships or clearly consists of subproblems?*

There might be problems that have a clear separation of subtasks the ANN need to perform, and it would be interesting to see if the algorithm is able to identify and solve these. Since the substrates are free to evolve, an interesting thing to investigate is whether the evolution will make substrates that reflect the problem it evolves to solve.

Research question 3 *How does evolved substrates perform on modular problems compared to manually designed substrates?*

Standard HyperNEAT assumes the user to design a substrate that reflects aspects of the problem being solved. It is therefore interesting to see if this manual task can be achieved automatically by evolution. If the spatial information in the evolved substrates are as good as hand-crafted substrates, it would suggest that evolution can be trusted with the substrate geometry. Since modular problems are difficult to solve with standard HyperNEAT, confirming whether these kinds of problems can be solved with substrate evolution could expand the possible application areas, and simplify the use of HyperNEAT.

1.3 Structured Literature Review Protocol

This section elaborates on the process in which the litterateur was researched and how the central thesis of the work was reached. This sections also presents the central inclusion criteria for what papers was included, as well as the quality criteria in which the value of research was assessed.

1.3.1 Literature Sources

The literature sources was mainly gathered from *ACM Genetic and Evolutionary Computation Conference* (GECCO), *IEEE Congress on Evolutionary Computation* (CEC). Importance was placed on recent publications from 2018 and 2017. Google Scholar was used to find cited papers, and for very specific searches in later part of the project.

1.3.2 Criteria for inclusion

These criteria were used for what papers that were included in the primary screening. This was mostly based on the paper's title and abstract:

ID	Inclusion Criteria
1	The study's main concern is evolving topology for ANN
2	The study is a primary study that includes empirical results
4	The study focuses on a possible solution for topology search
5	The study focuses on a challenge for finding good topologies

Table 1.1: Inclusion Criteria for Primary literature search.

1.3.3 Criteria for evaluation of papers

These criteria were used to further narrow down papers of interest, and assess their reliability:

ID	Quality Criteria
QC1	Report empirical results on at least two data sets
QC2	The study is put into context of other research and state of the art of the time?
QC4	Heavy Computational resources not required?
QC5	Are algorithm design justified?
QC6	There is a clear statement of the aim of the research
QC7	Implementation code published on web?
QC8	Reflected on how design decisions are reflected in results
QC9	The study is published in the last 3 years
QC10	Evolves connectivity-topology
QC11	Evolves neuron-topology

Table 1.2: Quality Criteria for assessing research.

1.4 Literature Research Phases

To arrive at the final thesis for the master project was a process that went through four phases. Through all these phases the focus was on genetic algorithms and topology search in ANN. These are big and broad subjects, and thus this was necessary to gain knowledge of earlier work already done, and to search for interesting topics not yet explored. This section goes through the four phases where the literature was explored from different angles. A graphic representation of the most important topics researched can also be seen on the flowchart 1.1

1.4.1 Initial Literature Research Phase

The project did not have a set description at the start, and thus the initial stage was to find an interesting topic, become oriented in current state of the respective scientific field, and find something to add to the scientific community. An initial interest was topology search in neural networks, and genetic algorithms. Thus, the first focus was using GA for topology search.

During the initial research it quickly became apparent that the current literature in topology search often had image classification as application area, or used it for comparisons with other algorithms. Reasons for this is that image classification is a popular application area, and there are multiple well established benchmark data sets. Therefore, including image classification into the search process was very useful to find the latest work. This lead me into the next phase where image classification was included.

1.4.2 Second Literature Research Phase

When researching in this stage, the focused were maintained on methods that used GA, and thus other methods such at particle swarm optimization and genetic programming were discarded. Topics that was necessary to familiarize with was *CNN*, *lamarckian vs darwinian evolution*, and the variations of GA that had been used. Amongst other things the research on CNN introduced the importance of keeping spatial information through coordCNN. This idea of incorporating spatial information into solutions would become relevant in later search.

The literature search revealed that using genetic algorithms for topology search had already been done. However, the literature was often using very simple GA in combination with substantial amount of computer resources. Using more complicated GA for evolving ANN to be used in image classification seemed to be relatively unexplored. Inspired by applying a more sophisticated GA to the problem, the next research stage focused on researching what more sophisticated GA does, and why certain properties are aimed for.

1.4.3 Third Literature Research Phase

There exist many variations on GAs beyond what had been done in the papers for topology search in ANN for images. Thus, the research focus in this phase turned more to how small differences in GA algorithms affected the population during evolution, and what properties seemed useful. Especially population diversity had been pointed out as beneficial in the papers oriented towards image classification, and so niching techniques were investigated. Modularity was also pointed out as a perceived useful trait in ANN. Two more sophisticated GA algorithms was

NEAT and HyperNEAT, where the former used direct encoding and the latter indirect encoding. Thus, the pros and cons of indirect encoding was researched.

NEAT is an GA algorithm for neuroevolution that was omnipresent in the related work sections for papers cornering neuroevolution, so it quickly attraction attention. In addition to having novel innovations, it also applied methods from the niching techniques. NEAT is an direct encoding, and already very well explored. In addition to NEAT, CPPNs and HyperNEAT was also very often mentioned. CPPNs being an ANN that is very relevant for images, and HyperNEAT being an indirect encoding for neruoevoution that cares about spatial information.

The indirect encoding, attention to spatial information and close inspiration from developmental evolution made HyperNEAT very interesting. With reflections on GA properties that was conducted in this phase, HyperNEAT seemed to have certain limitations. The literature search was thus narrowed down to focus on HyperNEAT

1.4.4 Final Literature Research Phase

Amongst the properties that had been investigated in the previous phase, modularity was one property that had been examined in relation to HyperNEAT. Multiple techniques for improving modularity in HyperNEAT had been published. A limitation that was identified however was that the substrate was excluded from evolution. Papers that discussed actually evolving the substrate was limited. This ended up with a final focus being on evolving substrates in HyperNEAT, and investigate if it could help HyperNEAT solve modular problems.

1.5 Thesis Structure

The following chapters in the thesis will consist of the following: Chapter 2 contains background theory relevant to the thesis as well as explaining terminology used in following chapters. Chapter 2 ends with a look at the state of the art, and puts this thesis in context of earlier work. Chapter 3 lays out the proposed extension to HyperNEAT. Chapter 4 contains the experimental plan to answer the research questions, as well as results obtained from those experiments. Chapter 5 contains the discussion and contributions, and ends with a look at possible future work.

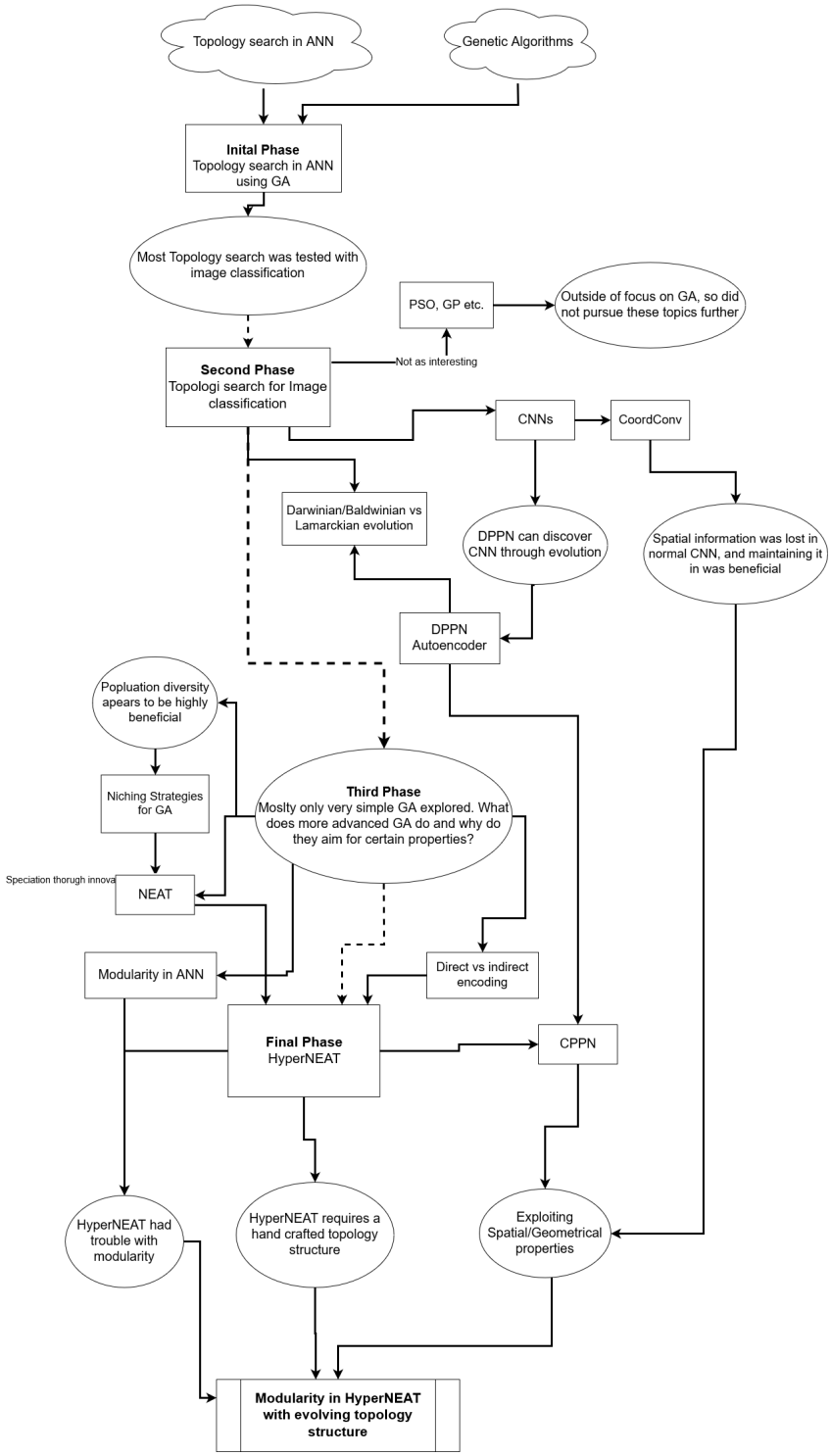


Figure 1.1: Flowchart on how the central thesis of the project was reached, and what literature was visited on the way.

Chapter 2

Background Theory and Motivation

This chapter elaborates on the necessary background theory the thesis is built upon, as well of the current state of the art. Then at the end follows a more in depth motivation in light of the previous work presented in this chapter.

2.1 Background Theory

The thesis build upon theory from neural networks, genetic algorithms and developmental evolution, and modularity. This section is aimed at introducing basic concepts as well as well established theory necessary for the thesis.

2.1.1 Artificial Neural Network (ANN)

An Artificial Neural Network is a computational system inspired by the human brain. However, all biological processes are abstracted away into neurons with ingoing and outgoing connections, and internal neuron activity is abstracted away into an activation function. Figure 2.1 depicts such an abstraction of one neuron receiving three signals and processing it to give an output. Each connection has a corresponding *weight* that are multiplied with the signal strength (x) from each signal. These products are summed up and then added to a *bias* that each neuron have. This value are then passed through an activation function depicted as f(y), and the output of this activation function will be the output value of the neuron.

ANNs are powerful data structures that can both model and learn complex relationships. Because of this ANNs are often used for optimization and classification tasks. ANNs can take many shapes, but this work will focus on feed-forward

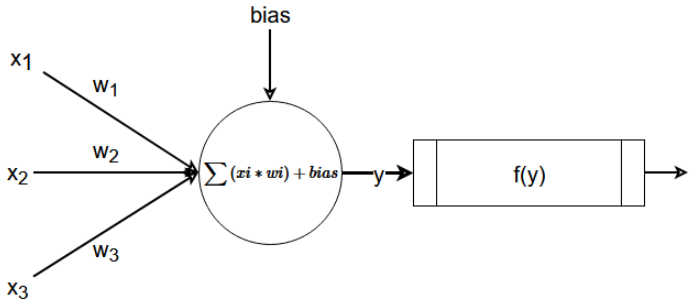


Figure 2.1: Abstraction of a neuron in an ANN

networks. In a feed-forward network all information flows one way, and the network don't have any memory. If a group of neurons are located so that they all can receive signal from the same neurons and can transmit to the same neurons, they are considered to be in the same *layer*, shown in Figure 2.2. In a feed forward ANN layers can't transmit to backwards to lower layers. Feed forward ANNs are popular because they are relatively easy to train, and fast in the sense that each artificial neuron only process information one time.

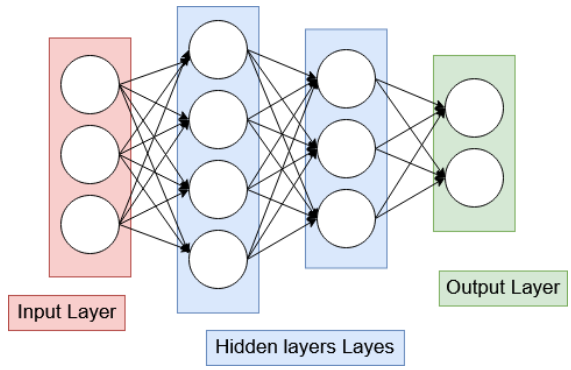


Figure 2.2: A depiction of ANN layers

The topology of an ANN have a big impact on its performance, and what is considered a good topology for an ANN is highly problem specific. Finding good

topology are thus considered an important non-trivial task. ANN always consist of two distinct types of neurons. Input neurons and output neurons. Nodes that are between these two are considered "hidden" neurons, because the user only interact with input and output neurons.

2.1.2 Genetic Algorithm (GA)

A genetic algorithm(GA) are population based evolutionary algorithms that evolve through *mutations* and sometimes *crossover*. Mutations are random mutations that happen to an individual to change it, and can by chance make new individuals in the population more fit. Crossover is the combination of genetic material from more than one individual, and good traits can spread through the population. The normal flow of an genetic algorithm is presented in Figure 2.3, where the main loop consist of parent selection, crossover, mutation, fitness evaluation and insertion of the new individuals into the population. After theses steps, the GA tests for stop conditions to see if it should continue or return the best solution found. Stop conditions are usually a performance criteria for the best solution, or a specified number of generations.

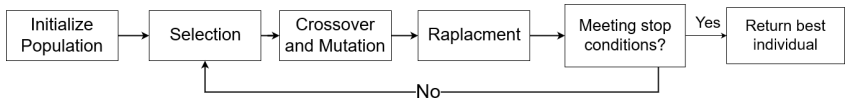


Figure 2.3: The main loop of a Genetic Algorithm

2.1.3 Modularity

Modularity is quite prevalent in biology. For example in metabolic networks[10], animal brains[20], protein-protein interactions, RNA[37] and body plans[4].

Earlier work about modularity in ANNs have separated modularity further into several distinct types of modularity, the most important distinction being *topological modularity*(T-modularity) and *functional modularity*(F-modularity)[37; 1]. T-modularity refers to how nodes and modules in a network connect to each other. F-modularity on the other hand refers to a separation of functionality between the modules. This is relevant to how the modules actually solve subproblems of an overall task. If for example a system with three processing units are to perform three different tasks, a natural choice is to have each processing unit handle each task autonomously. This is illustrated in Figure 2.4 a), where three color-coded tasks are processed separately and then gathered together afterwards. This is however not the only way to do it, an alternative can be seen

in b) where each processing unit handles 1/3 of all the task. b) would not be considered to have F-modularity since each tasks are calculated over multiple modules mixed in with other tasks. The fact that a network shows T-modularity, does not necessitate that the calculations for a task are contained within a module. Thus, T-modularity is necessary for F-modularity, but it does not by itself guarantee the presence of F-modularity. It has been stated that a T-modularity is not guaranteed to result in functional specialization in the modules unless there is a learning algorithm that promotes F-modularity[1]. F-modularity can be difficult to confirm exist given that ANNs are often viewed as a black box precisely because of how hard it is to gain an intuitive understanding of how a given ANN process information.

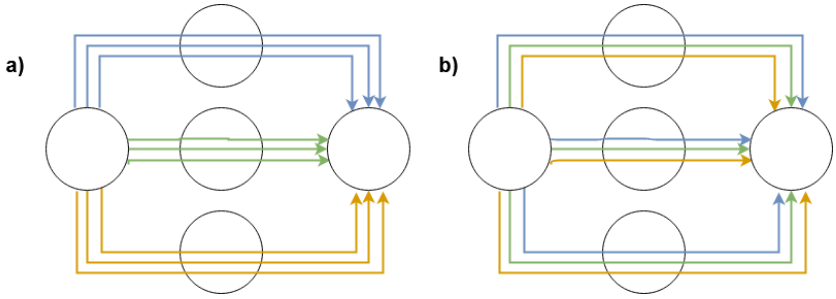


Figure 2.4: A depiction of how T-modularity does not have to result in F-modularity

There are many theories on why modularity appear to be so desired in nature. Modularity seem to offer several advantages, and there is probably not one, but a combination of these that have made it present in biological neural networks(BNN). Benefits for modularity that have been identified can generally be divided up into: variational adaption, pleiotrophic effects, avoiding spatial interference, constraints form physics, noise in genotype-phenotype mapping[12], and to lower catastrophic forgetting[7].

Variational adaption refers to when where multiple traits that needs to change at the same time as response to changing environmental pressure, are grouped together[37]. Likewise traits that rarely needs to be changed at the same time exist in separate modules. The source of the pressure here are thus external environmental changes, and modularity gives a benefit because it allows species to adjust faster to changing goals and environments.

Pleitropy is term that describes the genetic effect of one gene on multiple phe-

notype traits[30]. If one gene affects a lot of traits, there is an increased chance that a mutation will have an destructive effect and cripple the organism because the more phenotype traits one gene affect, the harder it is to adjust it without breaking existing functionality. If however the mutation was contained in one module, the organism might still be able to function. Both because the affect of destructive mutations are limited to one module, and because changes that are beneficial for one module, don't also negatively impact many other modules. Additionally this allows a module to change independently from the others. In essence; if a mutation turns out to benefit one module, it can't also be destructive for multiple other modules if it does not affect them. The pressure for modularity is thus apparent in the long run where it makes the evolutionary process more effective.

Spatial interference is a conflict between learning or evolving two opposing tasks at the same time[30]. Solutions for one task would be destructive for the other task. If on the other hand each task is solved in separate part of the system, spatial interference could be avoided and this gives a positive pressure for modularity. This conflict can occur both during evolution of the population, and during learning for each individual. Spatial interference can apply whenever a system changes to adjust to an environment, but the problem of spatial interference is usually focused learning in neural networks. The pressure thus comes from the ability to better adjust to external environments.

Modularity could also be a response to physical constraints. An good example would be in BNN where each connection have an energy cost, and this would give a pressure towards sparse networks. Not only would it give a pressure to create few connections, but pressure to many local short connections. This is because of a strong bias against many long connections, and a bias toward connectivity with the closest neurons so that the energy cost for the cell are low. From this it is not a long step towards many "short" connections between spatially close neurons, and a relative few longer connections that connects the module with the rest of the neural net. In this way optimizing for physical constraints, have an additional benefit of giving favourable conditions for T-modularity to occur. This theory are a bit separate from the others given that it does not state directly that modularity gives an evolutionary advantage, but rather that modularity is a natural consequence of optimizing for low energy costs. These physical constraints are not something we have to have in computers, but ANN have been shown to increase in modularity if these constraints are simulated[7].

2.1.4 NEAT

NeuroEvolution of Augmenting Topologies (NEAT)[34] is a genetic algorithm for evolving the topology and values in an ANN. NEAT accomplishes this by evolving a genome that consist both node genes, and weight genes. NEAT has the favourable property of starting with a small simple ANN, and over generations iteratively increase the number of nodes and connections. This allows the algorithm to adjust the complexity of the ANN to the necessary level for different problems. NEAT is thus a GA specifically designed to evolve ANN.

The main contribution that NEAT brought was the concept of *innovation numbers*. Genetic algorithms may use both mutations and crossovers, however crossovers have a risk of mixing genes for very different functions, and thus be very damaging to the new individual. In biology for example, if a crossover replaces genes crucial for heart functions, it would be very beneficial if the replacement genes can perform the same functionality. This is generally called the *competing convention problem* or the *permutations problem*, where important information can be lost due to crossover that "overwrites" useful genetic material. A popular example of the problem can be seen in Figure 2.5 where two ANN perform the same functions A,B and C, but it happens in diffrent parts of the network.

In an ANN it is difficult to know what part of the network performs what function, and so the NEAT makes an assumption based on common heritage. If genes have common origin, it is assumed that they fulfils somewhat the same function in the ANN. Innovation number is an unique number given to each gene at creation. This way if two genes from different individual are to be crossed over and they have they both have a gene with the same innovation number, NEAT will consider it relatively safe to swap these genes. This lowers the chances of new individuals being stillborn and this makes the GA more efficient.

NEAT also use the innovation numbers for *speciation*. Speciation is a *niching* technique for maintaining a diverse population, where individuals in the population are grouped into species decided by similarity. The best from each species are guarantied to survive to the next generation, preventing premature convergence by preventing one species to outcompete all others. This keeps variation in the population, and gives new solutions time to mature before they are discarded as they only need to compete against their own species. During evolution, species that have stops improving will eventually die off, and evolution will thus change focus from exploration to exploitation of the few remaining species. NEAT uses the innovation numbers in the genome to determine distance between individuals. If the values of genes sharing a common origin have diverged a lot, this would increase the distance between individuals having the different versions of that gene.

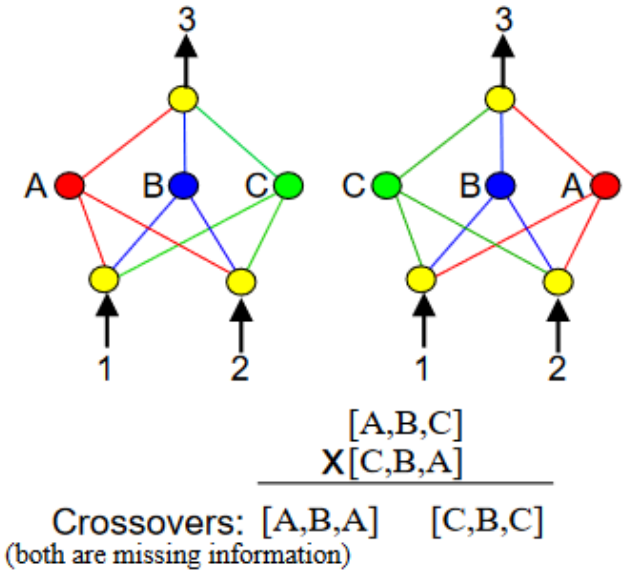


Figure 2.5: The Competing convention problem shown for two networks that perform the same functions in different neurons

The innovation numbers are unique identifiers for genes, and this can is used if genes need to relate to other genes. This is something the connection genes are dependent on being able to do, since they should always connect neurons originating from the same neuron-genes. To accomplish this it is possible to store this information in the innovation number of a connection gene. When this is done it is not an integer, but a tuple of containing two integers. One specifying what neuron the signal originates from, and one to identify the receiver neuron. If connection genes store the connections as values or in the innovation key depends on the implementation, and there is no clear conscientious on the best approach. Having an unique integer allows for mutations that occur separately to be separated, even if they connect the same genes. With a tuple as innovation key there are guarantied to never be more than one connection between two neurons, and connections between two neurons have a high likelihood of contributing to the same function and this would thus be reflected in the common innovation number even if the mutations happened at separate times in evolution.

A mentioned earlier, NEAT have genes for both nodes and connections. This

means that NEAT works on a representation that is a *direct encoding*, where each node-gene are translated to one node, and each weight gene translates to one weight in the phenotype ANN. Because NEAT have a one-to-one relationship between the size of the ANN and the genome. This has the unfortunate effect that if you want to optimize a network on the size of a human brain with 100 trillions connections, you need to optimize every single gene in a genome consisting of at least 100 trillion genes. For comparison the human genome are able to achieve this with only about 30 000 genes[8**?**]. For this reason, NEAT has some problems optimizing very large networks.

CPPN

A Compositional Pattern Producing Network(CPPN)[32] is an ANN with the goal of producing a wide array of possible patterns that has interesting or useful properties. CPPN have its inspiration from biological embryonic development, and CPPN were designed to make these patterns are because they can be combined to represent an abstraction of how chemical gradients interact during development. Since CPPN is an abstraction, it avoids having to simulate a growth period with local interactions.

The patterns that can be represented by a CPPN include symmetry, imperfect symmetry, repetition, asymmetry, variation, and repetition with variation. Some examples this can be seen in Figure 2.6, where pixel coordinates are used as input and output decided the pixel value. It illustrates that CPPN can abstract complex gradient interactions, with examples including: Symmetry(a), Imperfect Symmetry(b), as well as Receptions with Variation(c). All of these can be observed in biological organisms. Left right symmetry is observed in the brain and body plans of vertebrates, right handedness is an example of imperfect symmetry. Variation is seen in the brain's minicolumns in the neocortex[3].

The thing that separates CPPN from normal ANN is the set of possible activation functions. Neurons in a CPPN have activation functions well suited for patterns, for example: sigmoid, tanh, absolute value, Gaussian, identity and sine, and rectified linear units (ReLU). The exact set of possible activation functions vary depending on the implementation, but Gaussian is usually included for left right symmetry and sin is also often included for repetition. The CPPN can create complex patterns while consisting of relatively small network. This makes it well suited to be evolved with NEAT, something that was done in HyperNEAT.

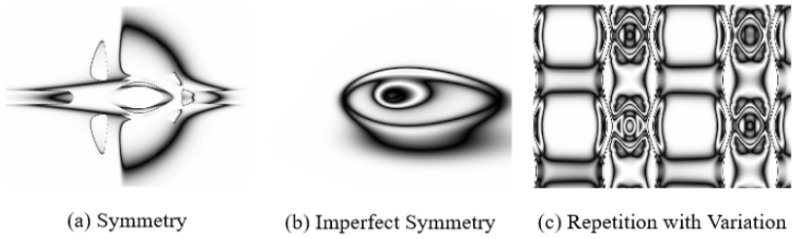


Figure 2.6: Depictions of how CPPN patterns can be used as an abstraction of complex gradient interactions

2.1.5 HyperNEAT

HyperNEAT is an algorithm which builds upon the NEAT and CPPN algorithms previously mentioned. Unlike NEAT, HyperNEAT does not have the connections represented as genes in the genome, but instead uses a CPPN evolved with NEAT to determine the connectivity patterns in an ANN. For this reason, HyperNEAT is considered to use *indirect encoding* for the connections. There are multiple advantages with indirect encoding. The use of CPPN allows the description of the solution to be compressed down and represented in a lower dimensional space. Additionally information can be reused, and thus the final ANN can be larger than the genome itself.

Building on from CPPN, HyperNEAT also take inspiration from biological development. In biology cells adapts their function depending on the chemical gradients experienced at the cell's geometric location. As the output of a CPPN describe an abstraction of interacting gradients, the neuron geometry are used as input to the CPPN, and the outputs are used for the connection values. HyperNEAT thus compute the network connectivity as a function of the neurons' geometry. To give all the neurons geometric coordinates, they are placed in a coordinate system called a *substrate*. It is normal to place the neuron in the substrate in a way that reflect the geometry of the problem, and this allows geometric relationships in a problem can be exploited.

Thus HyperNEAT consist of two entities, a substrate and a CPPN. The former shown at the left in Figure 2.7 and the latter shown to the right. If we want to assign a connection value between two neurons, their coordinates are given as input to the CPPN, and the CPPN outputs the weights that should be between the two nodes. If the value is above a certain threshold, a connection will be

formed. This can be seen in Figure 2.7 where a weight between two nodes in a substrate(left) is decided by the CPPN on the right. Substrates usually have two dimensions, and thus all neurons have X,Y coordinates. This gives a total of four inputs, and these are used as the input to the CPPN, and the CPPN output "W" decides the weight between the neurons. Notice also the multiple different activation functions in the CPPN. It is also normal to have a CPPN output node for bias (not depicted in Figure 2.7)

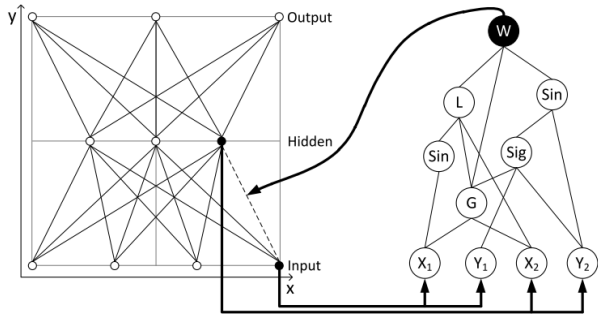


Figure 2.7: A weight in HyperNEAT are decided by inputting the neuron coordinates into the CPPN

In HyperNEAT, all nodes are typically located on the same substrate. This is also called the crowded substrate, because it squeezes several types of neurons into the same space, with the only "neuron group separation" being distance. An example of this is in Figure 2.7 where the only difference between different types of neurons are distance along the Y dimension. In the crowded substrate the CPPN will tend to give similar weights to nodes that are close to each other, because of similar CPPN inputs. This might not be a valid correlation and has the risk of placing the population in an early evolutionary dead end.

2.2 State of the Art

This section will present the current state of the art for techniques that increase modularity in HyperNEAT, and techniques for changing the substrate in HyperNEAT. Section 1 will go through the state of the art extensions to HyperNEAT designed to increase modularity. Section 2 goes through a technique where instead of using only one substrate, multiple are used. The chapter then ends on the two current state of the art methods for evolving substrates in HyperNEAT.

2.2.1 Modularity in HyperNEAT

As mentioned in section 1.1 and 2.1.3 biological evolution have selected for modularity in BNNs, yet HyperNEAT often don't create ANN that exhibit high modularity[6]. As a response to low modularity in the ANNs produced by HyperNEAT, Verbancsics and Stanley [36] proposed several techniques in which evolution could be biased towards modularity. One technique was the *Dynamic Distance Threshold*(DDT) that exploited the substrate geometry to determine threshold values. In DDT the distances between neurons determines the threshold value that are used. This leads to close neurons forming connections even though the weight values are weak, and connections between further apart neurons are only expressed if they have a very high value. Even though it did lead to modular ANNs, these ANNs showed poor performance. Because of this DDT have not received much further attention in the literature. Conversely one technique that has become quite popular is to allow evolution to learn connectivity separately from weight values. As stated in section 2.1.5, HyperNEAT uses a threshold for the weight values to determine the existence of a connection. In contrast, HyperNEAT with *Link-Expression Output*(LEO) has a dedicated CPPN output-node that only determines if a connection should exist or not. This gives the evolution more freedom to alter the connectivity pattern and the weight pattern independently. LEO can also easily be combined with other techniques for modularity.

LEO was also shown to work very well together with another technique called *Gaussian Seed*(GS) that utilize the geometry of the substrate for modularity. GS encourage strong connectivity between close neurons, and discourage a connection between neurons that are far apart in the substrate. This is achieved by seeding the initial population of CPPNs with a hidden layer of neurons with gaussian activation functions. The seeded neurons receive the coordinate difference as input, and gaussian is chosen as activation function because it outputs maximum value when the difference is zero, and decrease as the nodes are further apart. GS is only done in the beginning of evolution, and exist only to set evolution on the correct path. Evolution might remove gaussian neurons, and there are no further pressure or mechanism for modularity after evolution has started. A technique for maintaining a pressure towards modality throughout the entire evolutionary run is to add a cost to network connections[7]. Adding a cost to connections simulates physical constraints discussed in 1.1, and are a general approach that are easy to implement. Extending HyperNEAT with a cost per connection is called *Connection-Cost Technique*(CCT)[13] and have been shown to maintain modularity throughout the evolution, compared to GS. With CCT the connection cost are calculated as the sum of squared lengths of all connections in the ANN, and thus the geometry of the substrate are directly determining the physical constraints that leads to modularity. This is a bit sim-

ilar to DDT, but instead of using the distance between nodes to determine the threshold for connectivity, the CCT does not directly affect the connectivity of an individual at all. Instead CCT affects the fitness the individual receives in the genetic algorithm, affecting the evolutionary pressure rather than the shape of an individual. HyperNEAT with CCT uses a 25% chance of including the connection cost into the fitness comparison of individuals. This is done to provide a stronger bias towards focusing on performance. This stronger bias towards performance is also is also illustrated by when modularity increases the most during evolution. HyperNEAT with CCT have shown a tendency to first optimize for performance, and then increase modularity when no more performance gains were possible.

In summary all of the state of the art extensions to HyperNEAT that increases modularity make actively use of the geometry in the substrate to simulate physical constraints discussed in section 2.1. CCT put a cost on each connection, similar to the extra energy cost these would have had in biological brains. Gausian seed tries to bias the network towards neurons that are close together, and discourage long connections. Dynamic distance makes it harder to form a connection the longer away the neurons are, sort of similar to the constraints of BNN. However, the geometry of the substrates have been a manual design decision. There is a lack of knowledge about how these techniques would affect, and be affected by allowing the substrate to change and including it into the evolution.

2.2.2 Measuring Functional Modularity

ANNs are traditionally seen as black box models, and it can be difficult to get an intuitive understanding of how they arrive at an answer. Because of this the typical way of assessing if an algorithm can achieve ANNs with F-modularity is to see if it can find solutions that perform well on problems that are assumed to either require or largely benefit from a modular solution. The most used task to test modularity for ANNs produced by HyperNEAT are called the retina task, first introduced by Kashtan and Alon [15]. Two other tasks used measure modularity for ANN produced by HyperNEAT are the 5-XOR and H-XOR tasks[36; 7]. These problems consist a small number of simple sub problems. The small size of the problems makes them easier to visualise, but it will also become easier for a solution to simply remember an input-output mapping without actually having to solve the individual sub problems in individual modules.

Retina problem

The retina problem is a visual task that tries to measure if the network can create two modules. In the context of HyperNEAT this means having a left right symmetry. It is pretty well established as a problem for testing modularity[13;

15; 36; 14; 28]. There have however been shown that this task can be solved with fully connected neural networks[7; 15], so it does not require modularity, even though it benefits from it.

A set of mirrored images are split up into two sets and give the labels "left" and "right". The network is supposed to recognise when it is given a left and a right object. Classically these images are 2x2, so there are 8 inputs total. The setup of can be seen in Figure 2.8 This makes a total of 256 possible inputs. Some variations exist, for example to not have the left and right images be mirrored and not having the AND, but instead use two outputs for each left and right object[14]. The left and right images can also be combined with an XOR or OR function instead of the standard AND. These variations can be used if one wants to evolve modularity with variational adaption adaption discussed in section 2.1.3.

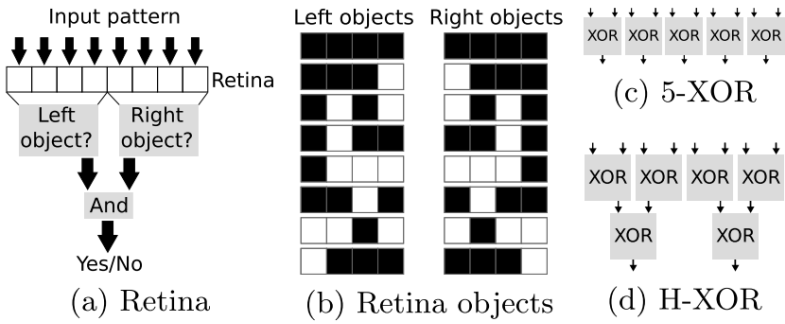


Figure 2.8: Depiction of problems requiring or benefiting from modularity, used to test for F-Modularity

5-XOR problem

The 5-XOR problem is a set of 5 XOR problems that needs to be solved in parallel (shown to the right in Figure 2.8). The problems has two inputs for each XOR task, so there is ten inputs total, and the ANN need to to provide 5 outputs. All the XOR problems are independent from another, and any connections between modules solving the different XOR sub-problems would therefore be detrimental to the ANN's performance. The ten different inputs allows for 1024 distinct inputs, so it is a larger problem than the retina task.

The 5-xor problems are well suited because it is a combination of non-linear tasks that are all similar. The similarity of all the xor-modules would be a benefit

for indirect encodings that are able to reuse functionality. Thus, maintaining HypernNEAT's ability for regularity and reuse is expected to be beneficial in this task.

Hierarchical XOR problem

The H-XOR is similar to the 5-XOR problem, but here the modules are required to interact. HXOR contains two modules where a pair of XOR feeds into another XOR. This problem has multiple levels of modularity, as depicted in Figure 2.8. Similar to Retina, it is divided into two parts, but these two parts have their own modularity with three xor problems in each. If an algorithm is able to create ANNs with separated modules, this task is well suited to investigate if the modules can effectively work together.

2.2.3 Measuring Topological Modularity

T-modularity does not involve how the ANN arrive at solutions to problems like with F-modularity, but rather it is decided by the structure of the neurons and connections that makes up the ANN. Because of this the ANN can be viewed as a connected graph. How to cluster nodes into modules with high connectivity within modules, and low connectivity out of the modules, is no easy task and it has been found to be NP complete[2]. A popular modularity measure is the Q score[22**?** ; 16; 23], where one tries to optimize groups of nodes so that each group has maximum internal connectivity, and minimize connections to other groups. The problem with this is that it only measure one hierarchical down. It does not care if modules also have sub-modules. The Q score are designed to be general for all networks, but this thesis are limited to feed forward ANNs and organized in layers, something that makes this easier. An approach to decompose a feed forward neural network into independent networks based on similar connection patterns has also been attempted[38].

These methods have origin in graph theory. Another angle of attack is to focus more on the signal interference of the back propagation signal, as was done by Soldal [30]. This method gives a score for modularity based on how much much overlap exist between the signal the output nodes receive. Formally this was be described as formula 2.1, where M is measure of modularity, N is number of hidden nodes, and N $\rightarrow$ o means how many hidden nodes influences output node o.

$$M = \frac{N}{\sum_{o=O}^{O} N- > o} \tag{2.1}$$

The modularity score is maximised if each hidden node only affect one output node, like depicted in Figure 2.9 a). If hidden nodes affect several different output

nodes, then the modularity score of the network goes down. If a new connection is made between two modules like in picture b), then two of the hidden nodes will contribute to both output nodes, mixing the two modules we saw in a). This results in a lower score for modularity. A fully connected network will have the lowest modularity score possible of N/N*O = 1/O. A benefit of this way of quantifying modularity is that it is relatively easy to calculate, and don't require a search for optimal Q score. A big reason for this is that it does not need to group the neurons into modules, but instead only measures the "overlap of influence" within the network. A limitation of this however is that it does not catch modularity that is "hidden" by other part of the network that might be more fully connected. An example of this can be seen in c), where the addition of two new hidden nodes results in all hidden nodes technically affecting all outputs, and thus giving it the lowest possible modularity score. At the same time, the two modules from a) are still clearly present in c), but it is still given the same score as a fully connected network. This limitation needs to be kept in mind when using this to measure T-modularity, as it will not detect modularity after a point where there is a fully connected layer, and it will not detect hierarchical modules where modules can consist of their own modules. This is sort of what happens in c) where all the hidden neurons are one module for the output neurons that comes after, but have their own internal modules. Thus, the current state of the art for measuring topology usually involves running another algorithm that optimizes for clusters to solve the NP problem, or exploit the feed forward nature to look at signal interference. This last one is a lot faster, but also has limitations in the types of modularities it can measure.

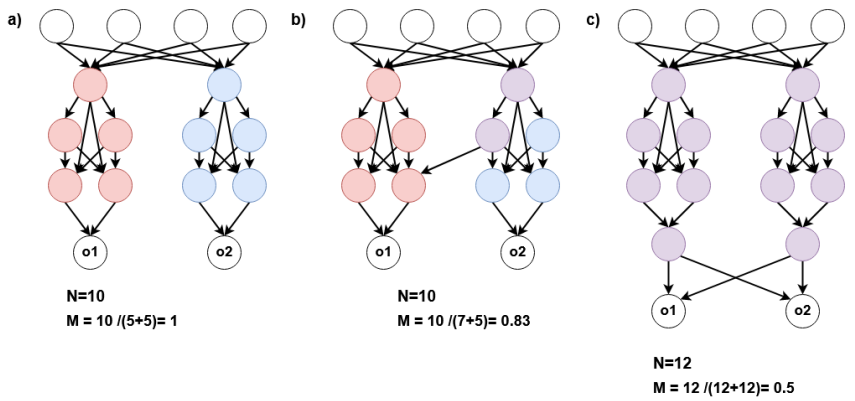


Figure 2.9: Examples on how modularity are calculated with Soldal's method on different modular networks.

2.2.4 HyperNEAT with Machine Learning

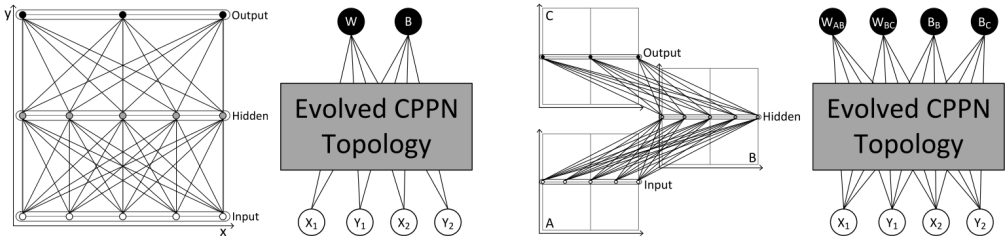
Back propagation have been shown to be able to improve HyperNEAT's performance on image classification tasks by Verbancsics and Harguess [35]. The substrate structure used consisted of four hidden layers, and all layers was connected to the two closest adjacent layers. When HyperNEAT was used together with BP it could use smaller networks compared to HyperNEAT without BP, and still achieve significant better classification accuracy. It was reported an increase from 23.9% to 58.4%. An even higher classification accuracy were achieved when HyperNEAT with BP were used to evolve CNNs. This achieved a classification accuracy of 92%. This work shows an example of how machine-learning can be combined with HyperNEAT for improved results.

2.2.5 Multiple Substrate Structure

An alternative to the crowded substrate is called Multi-Spatial Substrate(MSS)[25], where neurons in distinct groups can be placed in separate substrates. An example of this can be seen on the right in Figure 2.10 where input, output and hidden neurons no longer have to share the same substrate. This limits the risk of unrelated neurons having similar enough coordinates to cause unfortunate correlations early in evolution[25]. This is especially a risk when the number of nodes and number of hidden layers increases, since there will be more neurons having to occupy a substrate that stays the same size. In essence the MSS simplifies the substrate design since the designer no longer need decide on how to place unrelated groups of neurons together in the same substrate. Instead different groups of neurons can be divided up and placed in different substrates as shown in Figure 2.10. The left image shows a "classical HyperNEAT", and the right shows how it could be represented in a MSS. The hidden nodes are given their own substrate, separate from the input and output nodes who also have their own substrate. If there are multiple groups of hidden neurons, these can all be given their own independent substrate

To adapt the CPPN to a MSS, the CPPN are given distinct output nodes dedicated for each substrate pair, (Figure 2.10 d). In this way MSS allows distinct group of neurons to be completely separated, since the same inputs can result in different output for each CPPN output node. Because this is done, neurons that have similar positions, but in different substrates, can have completely different weights given to it from the CPPN, since we read from different output nodes. In this way the CPPN for MSS-HyperNEAT does not need to have one CPPN output that fit all the neurons, but the task are instead divided up amongst many CPPN outputs. One reason for not splitting it into several independent networks for each substrate, is that when they all go through the same CPPN, common

a



(a) Crowded substrate (CS) **(b)** CPPN (CS) **(c)** Multi-spatial substrate (MSS) **(d)** CPPN (MSS)

Figure 2.10: Depiction of the difference between crowded substrate and a Multi-Spatial Substrate, and how the CPPN output are adapted

general things can be handled higher up in the CPPN, before the output for each substrate pair is specialized towards the end of the CPPN. This allows for more reuse of the network, and for general global patterns to be formed shared by multiple substrates, at the same time as each substrate pair can get a specialised pattern.

On the flip side MSS have a lot more neurons in the CPPN compared to normal HyperNEAT, increasing the complexity. Since there is one new CPPN output for each possible pair of substrates that can connect, this can have problems scaling up to substrate-structures consisting of a large amount of substrates. If each substrate pair have its own CPPN output, we will have (S(S-1))/2 pairs for S substrates. In the worst case where it is fully connected, there will be O(S**2) CPPN outputs. This is illustrated by Figure 2.11 where only six substrates for hidden neurons placed in two layers can result in 22 sets of "weight, bias, LEO"-outputs (only 11 depicted). In total 66 output neurons for the CPPN to evolve. The design of the substrates becomes more easy with MSS, but a drawback is that the CPPN has a lot more outputs that evolution have find good solutions for.

At first MSS only used x and y from both neurons as input to the CPPN, making the input to the CPPN be (x1,y1,x2,y2). Other methods have included input that provides the CPPN with information of where the substrate is located in within substrate structure. This is done by including a z coordinate to represent the layer of the substrate[31]. Another solution is to add one more coordinate called s and have it represent the location of a substrate within the layer[29]. Thus, both the depth and width location of the substrate are provided to the

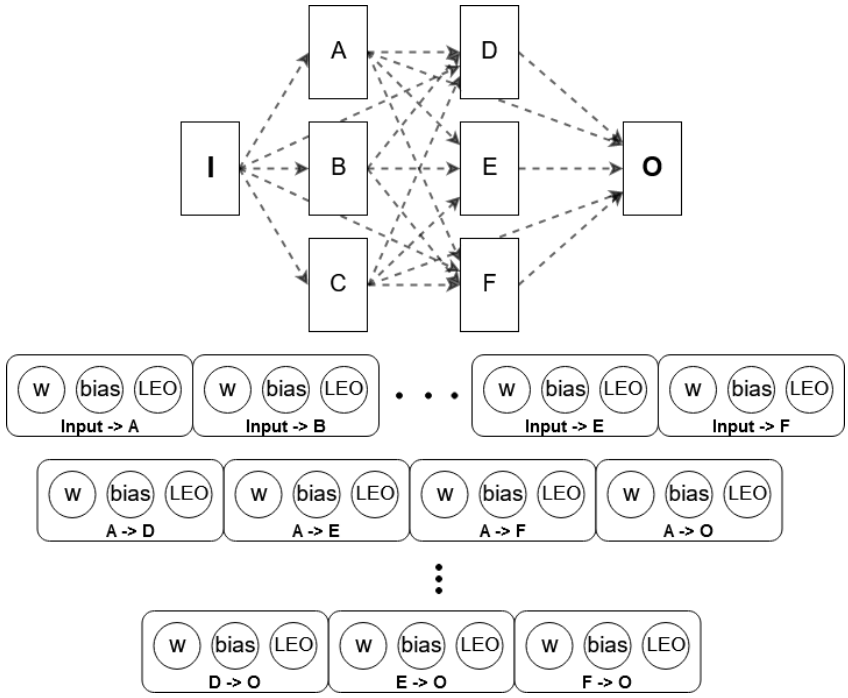


Figure 2.11: Example of MSS scaling with 6 hidden substrates resulting in 22 sets of CPPN outputs

CPPN. In addition to the x and y this gives four input for each neuron, resulting in eight inputs given to the CPPN for each neuron connection. A input to the CPPN would therefore be (x1,y1,s1,z1,x2,y2,s2,z2). These alternative methods for how to use CPPN with MSS allow for more spatial information to be given to the CPPN.

2.2.6 Evolving Substrates in HyperNEAT

Previous work on using HyperNEAT without an apriori substrate have been to have the substrate be dictated by the CPPN, so that only the CPPN needs to be evolved. This includes mainly *ES-HyperNEAT*[26], and *deepHyperNEAT*[31].

ES-HyperNEAT uses the CPPN output variance to find parts of the substrate to deactivate. This prunes away neurons in the substrate that are deemed unnecessary. If multiple nodes receive very similar connectivity from the CPPN,

they are assumed to fulfil the same function and can be merged into one neuron, shrinking the size of the ANN being produced. If for example multiple neurons close together are given the exact same input and output connections by the CPPN, ES-HyperNEAT will replace them with a single node. Which neurons are active in the substrate are thus decided only by the CPPN output, and the size of the substrate. This means that the substrate don't actually evolve independently, but is an indirect consequence of the CPPN. Thus, the substrate structure that will remain active is directly dependent on the variation of the CPPN output pattern. With this follows both pros and cons. The pros includes that indirect encoding allow for reuse, and less care needs to be taking when designing the substrate since the algorithm will prune away a lot of it. ES-HyperENAT was shown to be improved by LEO and GS[28], and it has also been extended to include local learning rules for improved plasticity[27]. Limitations on this method is that it assumes a-priori a maximum size of the substrate. The assumption on node redundancy build upon the assumption of all nodes having the same activation function. If activation function vary in the nodes, two nodes does not have to be performing the same function even though the CPPN assigns them both receive the same input and transmits to the same neurons.

Another approach to evolve substrates in HyperNEAT are the Deep-HyperNEAT, and it differs from ES-HyperNEAT by evolving a MSS instead of a crowded substrate, and does not utilize the same assumptions about CPPN output variation. Deep-HyperNEAT also don't starts off with a maximum size of the substrate or maximum number of neurons. Instead it grows the MSS in size and complexity more akin to how NEAT iterative evolve bigger and bigger ANN. This is done by decoding the CPPN into a substrate structure, by exploiting the fact that CPPNs for MSS-HyperNEAT have the substrates pairs marked on the CPPN outputs. By reading the CPPN outputs, one knows what substrates to include, and which substrates can have a connection. This is can be seen in Figure 2.12 where the labels of the blue outputs in the CPPN determines the number of substrates and the connectivity between them.

In Deep-HyperNEAT the CPPN can mutate to add new outputs, and thus also add new substrates, or new connections between substrates. The substrate structure can add substrates to increase both the depth and width of the ANN. This allows for the size and shape of the substrate structure to change, but each substrate in the substrate structure are essentially the same, and both the number and locations of nodes in the substrates don't change. This is a result of the shape of each substrate not being stored in the genome, and just the substrate's location and connections being decoded from the CPPN genome. This means that the shape of all the substrates are decided before the evolutionary run, but evolution

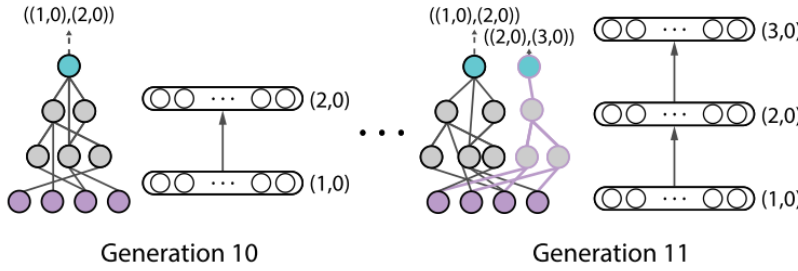


Figure 2.12: Example of how Deep-HyperNEAT CPPN output nodes determines the substrate structure, and thus changes with mutations to the CPPN

can use this shape as building blocks to build a substrate structure consisting of identical substrates.

2.2.7 Summary of State of the Art

HyperNEAT have been shown to be bad at creating modular networks and many of the state of the art HyperNEAT improvements have focused on this problem. Most of the modularity methods actively uses the structure of the substrate. The current state of the art work for changing the substrate structures have not allowed the substrate to evolve directly, but rather have it being decoded from a CPPN. This does not give evolution the opportunity of potentially trying multiple different substrate structure configurations with the same CPPN. The MSS-HyperNEAT introduced a noticeable change in how the substrate are organized and in how the CPPN is used, yet have not been used with a lot with the techniques for encouraging modularity. Given the current state of the art a natural next step to explore is to try to include the substrate structure into evolution, and combine this with methods created to utilize substrate geometries to make modular networks. Where earlier work have indirectly evolved a crowded substrate or a structure of inflexible substrates, this work aims at combining an evolving substrates in a MSS structure with methods for modularity.

Chapter 3

Algorithmic Model

This chapter goes into detail of the algorithmic model developed, how it was implemented, and design decisions taken. section 3.1 will give a short overview, before section 3.2 goes over how the substrate structure was represented in the evolutionary algorithm. Section 3.3 describes how the substrate evolves in the GA, section 3.4 presents how the genome is transformed into an ANN phenotype. Section 3.5 described how the ANN is trained and evaluated, followed by an description of possible CPPN variations in section 3.6. Finally section 3.7 will provide some details of implementations and design decisions.

3.1 Overview of Evolving MSS-HyperNEAT

The model consist of a combination of MSS-HyperNEAT and an evolutionary algorithm to evolve the substrate. The algorithm evolves genomes consisting of both CPPN-genes and Substrate-genes, and the entire genome is evolved by the same genetic algorithm. Having the entire genome be evolved by one algorithm limits the complexity compared too the added complexity if multiple different GAs were used. To accomplish this the NEAT algorithm is generalized to also handle arbitrary genes, so that the genes don't have to be only node-genes or connection-genes. This also allows for innovation number, speciation, disabling of genes, and tournament selection to be used when evolving the Substrate genes. To make it work with NEAT it was also necessary to represent the substrate structure as a direct encoding where each gene corresponds to one expressed trait in the phenotype. Since the CPPN in HyperNEAT is a direct encoding this a natural way of expanding the algorithm. Thus, both the CPPN and the substrate structure have direct encoding, and the connectivity pattern is the consequence of the interaction between these two, making the connectivity an

indirect encoding.

3.2 Representation of Substrates

The genes for the Substrate-genes consist of three different types of genes. These can be seen in Figure 3.1 where there are: layer, substrate and neuron genes, all gathered up into a dedicated *chromosome*. Because each gene mutates independently from the rest of the existing genome, they exist independently of order in the genome. Because they are gathered into layers and substrates upon decoding of the genome, there is a need to know how the genes relates to each. All neurons should be placed in the correct substrate and all substrates needs to be placed in the correct layer. The way his is achieved is similar to the possible implementation of connection genes in NEAT that was mentioned in section 2.1.4. Instead of using the innovation numbers to identify what neurons to connect, they are instead used to connect different gen types together. This a very practical solution since all genes in the genome already has an unique innovation key given to it by the NEAT algorithm. For the purpose of combining the genes, all substrate genes' innovation keys contain the innovation key of a layer gene, and all neuron's innovation keys contain the innovation key of a substrate. This can be seen in the top row in Table 3.2 where the bold subset of the innovation keys indicate where the gene should be placed upon decoding. For example the second gene from the left gene with innovation number [3,**0**] are a substrate gene that upon decoding will be placed in layer 0. Similarly, the neurons (0,[**3,0**]) and (1,[**3,0**]) will be placed in substrate [3,0]. To accomplish this the innovation keys of the substrate and neuron genes are not integers, but tuples where the second element are the innovation key of the "higher order structure" it belongs inside. Since layers are the highest structure they don't have tuples as innovating keys, but simply an integer.

After the genome are organized so that the correct neurons are grouped with the correct substrate, and the substrates are grouped with the correct layers, the substrate structure can be decoded from the genome. The hidden layers are organized after increasing innovation numbers, since higher innovation number means that it is newer. This is done so that each added layer can utilize all processing developed so far in evolution. It is also easy to implement. A possible alternative could have been to do it as NEAT, and simply not have defined layers, but since this is perceived as spatial information that could be useful for the CPPN, this was discarded. The innovation numbers are only used for the genetic evolution and organizing the genes, and are therefore not needed after the genome are decoded into a substrate structure and a CPPN. Instead the layers in the decoded substrate structure are identified by the ordering, starting

Layer Chromosome

Innovation number	0	1
Attributes	Enabled=True	Enabled=True

Substrate Chromosome

Innovation number	[3,0]	[5,1]
Attributes	Act.func=relu Enabled=True	Act.func=relu Enabled=True

Neruon Chromosome

Innovation number	(0,[3,0])	(1,[3,0])	(0,[5,1])
Attributes	X=1 Y=0 Enabled=True	X=0 Y=0.5 Enabled=True	X=0.5 Y=0.5 Enabled = True

Table 3.1: An example of chromosomes with different types of genes for the substrate structure.

Innovation number	0	[3,0]	(0,[3,0])	(1,[3,0])	1	[5,1]	(0,[5,1])
Gene type	Layer	Substrate	Neuron	Neuron	Layer	Substrate	Neuron
Attributes	Enabled=True	Act.func=relu Enabled=True	X=1 Y=0 Enabled=True	X=0 Y=0.5 Enabled=True	Enabled=True	Act.func=relu Enabled=True	X=0.5 Y=0.5 Enabled = True

Table 3.2: The relevant genes from Table 3.1 are gathered together before decoding

with input layer as 0 and ending with output as the higher layer. For example if a genome would have had layer genes with innovation keys: 0,1,7,10,12, it produces a substrate structure with layers: 0,1,2,3,4. Similarly, the substrates within a layer are simply a numbering from 0 to number of substrates if the layer, also called the *substrate order*. Discarding the innovation numbers once the substrate structure is created is also beneficial because it is used as input to the CPPN, where it is better to use coordinates 0, 1, 2, 3 rather than 0,3,18,64 for layer coordinates since the innovation numbers don't represent distance between the layers. For this reason, the green substrate gene [3,1] from Table 3.2 are turned into substrate (0,1) in Figure 3.1 because it is the first substrate in layer 1. The neuron that belong to this substrate have attributes X=0.5 and Y=0.5

and these are used to place the neuron within this substrate in Figure 3.1.

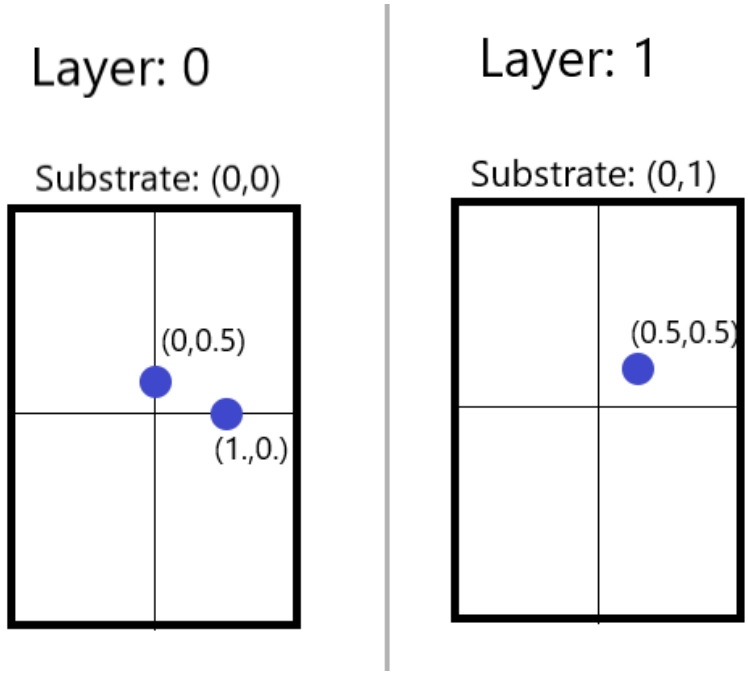


Figure 3.1: Genes from 3.2 would be decoded into this substrate structure with two substrates.

3.3 Evolving substrate genes

Evolution of the genome are done very similar to HyperNEAT, since it also uses NEAT. The main difference is that in HyperNEAT the NEAT only evolves CPPN genes, while in EMSS-HyperNEAT it also evolves the genes for the substrate structure, and their attributes. The new attributes introduced are the x,y coordinates and the activation function. These are evolved in a similar fashion as values are mutated in NEAT genomes. Except for a bit custom mutations, there are relatively few changes that needed to be done to a NEAT algorithm to make it into a general GA that can also evolve the genes for the substrate structure. Mutations that add layer genes will be able to expand the depth of the ANN by adding new layers, and mutations that add new substrates will expand the

width of existing layers in the ANN. This way both the depth and the width of the Substrate structure can incrementally increase during evolution. Depending on what CPPN adaption is in use, the creation and deletion of substrates might also create or remove genes related to CPPN outputs.

It is not desirable that the genome is filled up with layers and substrates that can't be expressed because they don't contain any neurons. To avoid this, all new layers are created with a corresponding substrate, and new substrates also creates some new neurons to fill it. If the last neuron of a substrate is deleted then the substrate deleted as well, and the same happens to a layer if if it no longer contains any substrates. The input and output layers are protected from deletion or being disabled, since it is assumed that all of them are will be useful for the ANN. Other than this the genetic evolution for the substrate genes are treated the same as other genes in NEAT.

3.4 Development of a Genome to an ANN

The development part is the intermediary steps where the genotype is turned into the a phenotype. This developmental part consists of combining genes into a substrate structure and a CPPN, and from these two parts create the actual ANN which would be the phenotype that is evaluated in the GA. An overview of this can be seen in Figure 3.2.

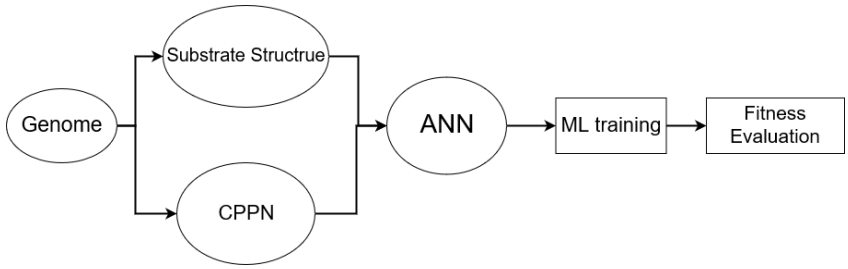


Figure 3.2: An overview of the process where a genome is developed into a ANN phenotype

The first thing that is done is to order the substrate genes and form layers of substrates as described in section 3.2. Then the CPPN is created from the CPPN node and connection genes. Building on from the MSS described in section 2.2.5,

this model uses eight inputs to the CPPN. This is done so that the CPPN can receive detailed information of a substrate's location within the substrate structure. This extra information is will probably be useful as the substrate structures might vary a lot during the evolution. Thus, the CPPN receive an *input vector* **(x1,y1,s1,z1,x2,y2,s2,z2)**, where **x** and **y** is coordinates for each node, **s** are the substrate order, and **z** is the depth of the layer. To better separate between s coordinates, all substrate orders are normalized between -1 and 1. This is done because many of the activation functions used vary most in this range, and it will make the s inputs more distinct and easier for the CPPN to separate.

After both the CPPN and Substrate structure are created from the genome, the CPPN are used to determine the connectivity between substrates. The ANN are constricted to be feed forward, and for this reason only connections to deeper layers are considered. For all the connections that are considered, the coordinates from the transmitting and the receiving substrate are used as input to the CPPN, and a LEO output node from the CPPN determines if a connection should exist or not. The LEO technique is used in EMSS-HyperNEAT because it allows for evolution to evolve the topology separately from the weight values, as discussed in section 2.2.1. Because connectivity are determined between substrates and not individual neurons as in standard HyperNEAT, x and y coordinates are not used in this stage. Instead only the s, and z coordinates are used since they provide the substrate's location within the substrate structure. After all coordinate input vectors have been send through the CPPN, all the connections between substrates are determined. This does not yet set the weights and biases in the ANN, but only what substrates transmits to what other substrate.

When a substrate transmits to another substrate it means that all neurons in the transmitting substrate transmits to all neurons in the receiving substrate. This was represented as a *connectivity matrix* where each row indicate where a substrate should transmit. The columns thus represents where a substrate receive signals from. A quick example can be seen in the matrix 3.4 where 1 indicate that a connection exist, and 0 that there is no connection between the substrates. From matrix it is possible to read that substrate 0 transmits to substrate 1, substrate 1 transmits to both substrate 2 and 3, substrate 2 transmits only to substrate 3, and substrate 3 is the output so it does not transmit to anything and thus the row only contain zeroes.

This matrix is a very useful substructure to have at this point in the developmental phase. It makes it easy to track how the signal propagates through the network. This can for example be used to check if the signals from the input substrate actually reaches the output substrate. The matrix quickly allows one to see if any substrate don't receive any signal by looking at the column and check-

Substrate ID	0	1	2	3
Substrate ID				
0	0	1	0	0
1	0	0	1	1
2	0	0	0	1
3	0	0	0	0

Table 3.3: An example of a connectivity matrix for four substrates

ing if it is all zero. Similarly, one can quickly check if any substrate transmits to any other substrate by simply check if any row only contains zeroes. This would mean that the substrate doesn't contribute to the network and might as well be removed (if it is not the output substrate). These scenarios can be addressed before any weight and bias values are given.

If a result of the connection matrix show that signal from the input substrate never flow to the output neurons there are several possible options. One solution is to simply give it fitness 0, and discard it. This would discard a lot of individuals, but also force the population to only have individuals where each substrate are connected to another by the CPPN. Since the fitness can be decided on this stage, no fitness evaluation has to be done on the individual. This is often what takes the majority of time in GAs. This would mitigate the computational cost somewhat, but not entirely. It could result in large parts of the new individuals each generation simply being thrown away.

Another solution is to somehow make sure that each substrate gets at least one input from another substrate. This avoids the large amount of individuals that are dead on arrival. Two options here are to connect it to all others, or simply to one substrate in the previous layer. These default options will be prevalent when CPPN don't connect it to anything, and there is a danger evolution can come to simply rely on the default, and not evolve the CPPN to connect specific substrates. If default set to connect to previous, this would give minimal connection cost if CPPN only outputted 0. This would mean that the CCT discussed in section 3.5.1 would enforce use of the default, since it would make the network as sparse as possible. Since modular networks should have more than one pathway through the network, this is not ideal. The option where a substrate without any inputs are given full connectivity would be strongly punished by the CCT. In this scenario the CCT would to give a benefits to CPPNs that can maintain few connections. This solution should result in CCT punishing any individual that relay on the default, while on the same time not having to spend time on

stillborn new individuals each generation. Because of this it become the method that was chosen.

Another thing that can be handled once the connection matrix is created is the scenario that arises whenever a substrate don't transmit any signal deeper in the network. This means that the substrate don't impact the output signal of the ANN. The easiest is to simply let it be there, since it does not affect the outputs. If they are to be expressed they function as Vestigial organs, in other words they don't have any impact on the phenotype. Pruning them away would make the networks easier to interpret, since they would not be cluttered by unused parts, but this alone was not enough to justify the small added computational cost. However, they should not be included in the calculation of connection costs, since connectivity between parts of the network that are irrelevant, should not impact the connection cost. If they were not pruned away, then they would be included into the maximum possible connectivity, and skew evolution towards individuals with large parts of the network unused and unconnected. For this reason all substrates that don't connect between input and output are removed.

After the connectivity between the substrates are decided, the weights and biases of the ANN is created. Each neuron in every substrate needs to have bias, and every connection between nodes in connected substrates needs a weight value. To find the weight values for all connections in the ANN, the coordinates of all transmitting and receiving neurons are used to create the CPPN input vectors **(x1,y1,s1,z1,x2,y2,s2,z2)** discussed earlier. If for example the neurons seen in Figure 4.5 were connected, then the input vectors for the two weights would be (x1=0, y1=0.5, s1=0, z1=0, x2=0.5, y2=0.5, s2=0, z2=1) and (x1=1, y1=0, s1=0, z1=0, x2=0.5, y2=0.5, s2=0, z2=1). For each input vector the weight between two neurons is outputted from the CPPN output node dedicated to weight values.

To decide the biases for all neurons in the ANN, one input vector for each neuron are created. These have the first half of the input vector filled with zeroes, other half is filled with the neuron's x,y,s and z coordinates. For example the neuron depicted in Layer 1 in Figure 4.5 would result in a CPPN vector that is (x1=0, y1=0, s=0, z=0, x2= 0.5 y2= 0.5, s=0, z=1). To set the bias, the dedicated CPPN output for bias is used. This is a lot of input vectors to run through the CPPN, but multiple can be run through the CPPN at the same time so it is not a problem. More details on what CPPN output that are used for a given input will be discussed more in 3.6.

At this stage the phenotype are a functioning ANN with connectivity and initial values. This is the end of the developmental phase and the CPPN and

substrate structure are no longer needed. At this stage the ANN is ready to be trained with machine learning if appropriate training data is available for the problem. For problems where this is not available, fitness evaluation would have to be done after the CPPN sets the initial values. The EMSS-HyperNEAT algorithm is designed with supervised learning in mind, but it can also be used for reinforcement learning if the machine learning is not used.

3.5 Training and Performance evaluation

After an ANN phenotype is produced from the genotype, machine learning can be used before the fitness evaluation. This is done by back propagation(BP) where the ANN output are compared to correct answers *labels* and any error are used to adjust the weights in the network. This does not change the existence of any connections, but instead the values of existing weights are adjusted. One of the benefits of modularity is that it avoids spatial interference as discussed in section 2.1.3. This effect should be stronger when the ANNs are trained versus only when it is evolving, because there will be a stronger conflict for whenever a value has to be set to two very different values for different training examples. In modular networks with low spatial interference the training should be a strong benefit, while ANNs that are not modular and thus have high spatial interference would have more trouble with training, since each training example will push the value in a different directions. As discussed earlier in section 2.1.3, even if a T-modular substrate structure is developed, that does not mean that the resulting ANN will have F-modularity. It is anticipated that having a learning algorithm that are hurt by spatial interference will promote the modules to specialize and solve tasks separately, and thus F-modularity is achieved. BP is also a very good technique for improving feed forward ANNs in general, so including this should make the algorithm more general and potentially able to handle a wider range of problems outside of specifically modular problems which the algorithm is mainly designed for. Since advancing a particular BP is not the focus of the project, the popular Adam algorithm[18] was chosen for optimization and the simple mean square error was used to calculate loss.

By including BP the model does not technically optimize for the best ANN to solve the problem, but instead the ANN is optimized for a topology and initial conditions that can effectively can utilize BP within the training time allowed per generation. This can be easier to achieve than optimizing for the exact ANN that will solve the problem.

Since the weights that results from training are not included in the genome, it is not passed to the next generation, same as in Darwinian evolution. Because the topology is a function of how two directly encoded structures interact (substrate

structure and CPPN), the weights can't be stored in the genome. For this reason it would not be possible to apply Lamarckian evolution where learned values are passed down to offspring.

During BP training the performance is measured four times on the validation set as depicted in Figure 3.3. The average of the four validation checks are then used as the performance of an individual in the fitness function. Taken four measurement of performance are done for two reasons: favouring quick learning individuals and an more stable performance evaluation. Since training takes time, the model is designed to creates a pressure to quickly converge on good solutions. This is more important than when only machine learning is used, since when it is combined into an evolutionary algorithm the resources has to be divided between all individuals in the population. The EMSS-HyperNEAT algorithm is designed so that if when a comparison happens between two different ANN that achieved the same accuracy at the end of training, the one that uses the least amount of computationally resources to reach that point will be favoured. The method used to calculate the performance creates an incentive to select individuals that can converge on high accuracy already in the first couple of evaluations, and at the same time favouring those that reach higher values at the end of the training.

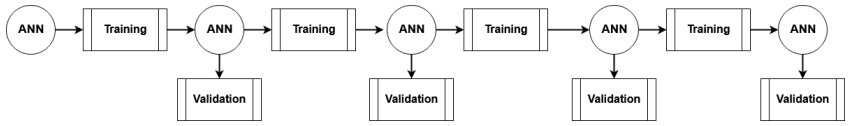


Figure 3.3: Validation are done at regular internals during the training period

If two individuals both reach the same accuracy, then the one that converged the fastest will be favoured. For example:

$$mean([0.2, 0.5, 0.8, 0.87]) = 0.592$$

$$mean([0.4, 0.75, 0.86, 0.87]) = 0.72$$

If two individuals both quickly reach a high accuracy, the one what continuous to improve the most will be favoured. For example:

$$mean([0.8, 0.81, 0.82, 0.82]) = 0.812$$

$$mean([0.8, 0.83, 0.85, 0.86]) = 0.835$$

A danger with this method of calculating the performance is that an individual that gets very high accuracy early, but stops improving might still get an higher

average score than individuals that improve more slowly but have not converged yet when the training time ends. An example of this is depicted in Figure 3.4, where given the shape of the fitness points, one would expect the lower graph to keep increasing if more training was applied. The use of the mean function will however favour the top graph even though it looks to be converging and little gain are expected from training it longer.

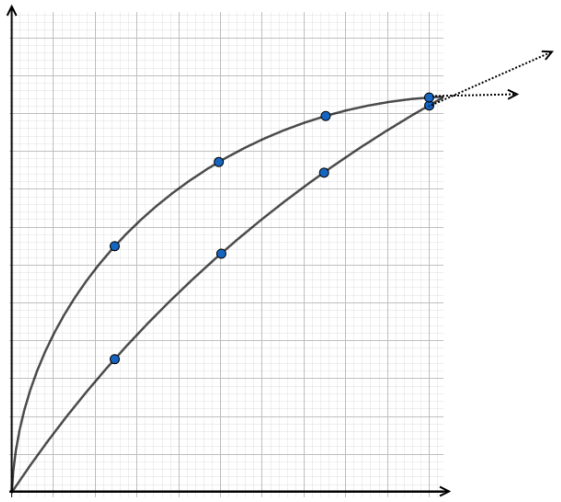


Figure 3.4: Depiction how the future expected performance gains would encourage ANNs that were still improving when training ended

Alternatives to combat this problem include giving more weight to the later evaluations to stronger discourage premature convergence, calculate the tangent for the last two points or somehow extrapolate the expected future fitness curve, and use the steepness to estimate future gains. To keep the complexity of the algorithm low and because of time constraints, the simple mean function was used instead.

The second reason for taking the average of four evaluating evaluations is that the ANN may not achieve the same accuracy each time after training. The reason for this is that BP training introduces some randomness because that training data are given in a random order. If performance was simply the accuracy the ANN accomplished at the end of the training period, the same ANN would have higher variation in the achieved accuracy at the end of training each time, leading to very inconsistent fitness scores. If each individual in the population have very

volatile fitness scores it will be difficult to evolve individuals that consistently perform well and it would encourage individuals that are volatile and inconsistent but have a chance of being lucky with the order of training examples. An obvious solution to this is to train the ANN multiple times and take the average, but this would require a lot more computational resources. Running the BP training a little longer and measuring accuracy at regular intervals does not have the same high additional computational cost. Thus, to make the fitness evaluation more stable, it was chosen to take the average of multiple accuracy achieved at regular intervals of the training. To sum up the aim of evaluating performance in this manner is to make the performance evaluations more stable, differentiate between individuals that achieve very similar performance, and encourage ANNs that require less training.

3.5.1 Connection Cost

The original CCT technique described in Section 2.1 worked poorly when applied to an evolving substrate. This is because unlike the original use of CCT in HyperNEAT, EMSS-HyperNEAT allows the maximum amount of connections to vary. To use the sum of squared length of all active connections in this model, will simply encourage evolution to design small networks with few possible connections. To avoid this effect, a new method for calculating Connection Cost(CC) was developed.

In this method the cost for connection length is instead calculated as the total sum of all the network's connection lengths, divided on the potential total sum of connectivity lengths it would have had if it was fully connected. This is done so that evolution is rewarded for sparsity in the connectivity, and not for the number of connections, since smaller networks automatically results in few connections. This is a way of getting sparse connectivity by simulating physical constraints as discussed in section 2.1.3. An example of this can be seen in Figure **??** where the total Connection Length Cost(CLC) for a fully connected network is 1, and the sparser the network is the lower the CLC will be.

Connection distance between two neurons are a function of the coordinates within the substrates space (x,y in Figure 3.6), and the distance between the layers (z in Figure 3.6). B' represent B in the plane of A's substrate in the figure, and Δx and Δy is simply the distance between A and B' in each dimension. The distance Δz represents how many layers the two substrates are apart. This design is intended to cause nodes that are connected to be rewarded for begin close, and also discourage connections between substrates in layers that are far apart. As

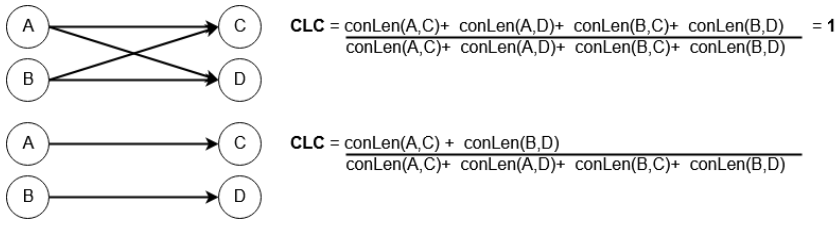


Figure 3.5: Connection Length Cost is sum of all connection lengths divided on the connection length for fully connected network of the same size

discussed in section 2.1.3, this can encourage modular ANNs. For example if a connection goes from layer one to layer number 34, it would get a very large Δ z, and connections between these two would result in a very long and punishing connection length. The connection length is formally calculated as:

$$ConnectionLenght(a, b) = \sqrt{\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}$$

To calculate CLC in this way have some advantages. If a new node is added, and it is fully connected to everything, the connection cost will not change since the same is added to actual connection length and to potential maximum connection length. This is shown in on the left in Figure 3.7 where a new node E is added. Since it is fully connected the same is added both above and under the division line. If the new node however is sparsely connected like depicted to the right in Figure 3.7, then the potential maximum will go up, but the actually connection length will only increase a little.

Thus the CLC will reward the innovations that are sparsely connected to the old network. This is advantages because new substrates that were fully connected to the old network would not be able to be part a separate discrete module. By encouraging new substrates that connects to less of the old network, it is more likely that it will not combine many separate modules, and instead join into a more independent module. The this way of calculating CLC neither encourage or discourage the actually size of the network, simply the ratio of connection length utilized. It is maximum at fully connected and minimum when the fewest possible connections are used. This should encourage modularity, and discourage mutations that add connections that results in mixing of the modules that are already discovered.

To calculate CLC in this way also have a disadvantage. It gives the lowest cost when there is only one connection between each layer. This is a network with maximum sparsity, since it each substrate only have one input and one output,

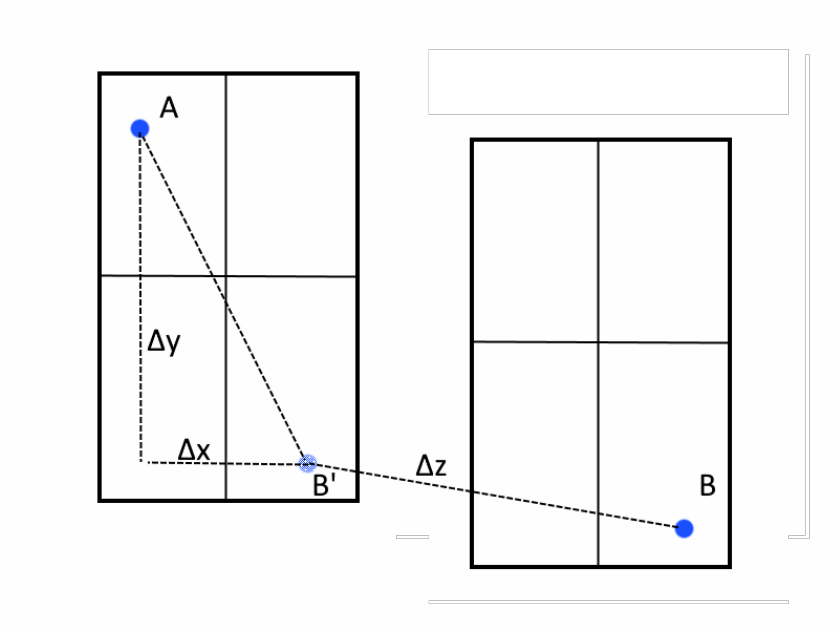


Figure 3.6: Connection length is a function of coordinates x,y,z. B' is B moved to A's substrate for clarity. Δz is how many layers B is away from A.

and therefore has the fewest possible connections. This problem is illustrated in Figure 3.8, where the network only uses one connection between each layer. If any of the red dotted connections would be added it would increase the connection length and thus increase CLC. This scenario is not favourable because the ANN cant have modularity if there is only one information pathway through the network. Even though modularity have been shown to occur in ANN by simply encouraging sparse networks, selecting for this to heavily might cripple the network. This may also be a reason why earlier work with simulating physical constraints in ANN applied CC in only 25% of generations as mentioned in 2.2.1.

To avoid optimizing for the scenario seen in Figure 3.8, a method for quantifying the degree of separate flows of information through the ANN were developed. The method developed is similar to how T-modularity was measured by Soldal [30], but instead of only measuring how much many hidden neurons affects each output neuron, it is done for all the hidden substrates and then averaged. This change will mitigate some of the limitations discussed in **??**, most notable mod-

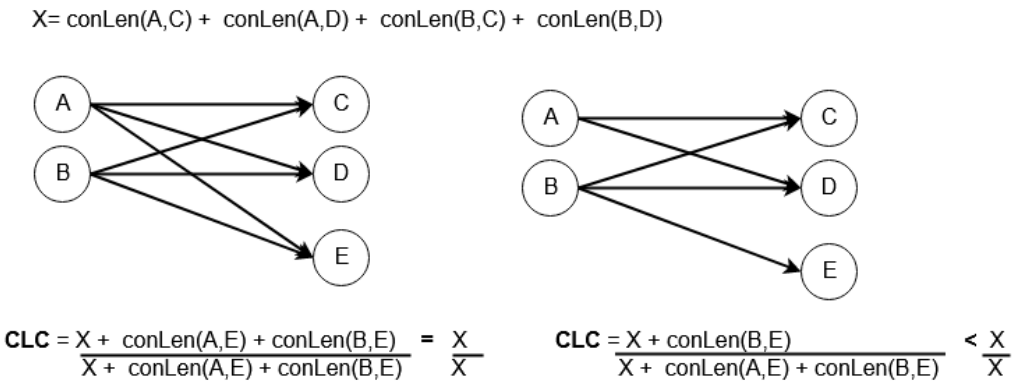


Figure 3.7: New additions to the network will not affect the CLC if it is fully connected, and lower the CLC if it results in a sparser network.

ularity will not be completely concealed by fully connected regions. A module typically have many connections within itself, and few to other modules. For a feed forward network this means that modules are not affected by the entire network, but only by a selected few which is part of that module. Thus, by measuring how big part of the network affect each substrate, a measure can be made for how interconnected the network is. In a a fully connected ANN without any modules all parts of the network would be influenced by 100% of earlier layers. In a modular network this should be lower, and the less interconnected the network is overall, the more separate pathways exist where modules within the ANN can work independently from the rest of the network. This measure of topological modularity are defined as :

$$Interference = \sum_{i=2}^{S}(\frac{s \rightarrow Si}{s})/S$$

Where S is the total number of hidden substrates. s is all upstream substrates before hidden substrate Si, and s $\rightarrow$ Si is the number of substrates that have whose signal reaches Si. An example are shown in Figure 3.9, where the top substrate in Layer 2 are effected by two substrates(colored blue), but could potentially receive signal from four(red). This is calculated for all the relevant substrates and then averaged.

All the input neurons are gathered together in one substrate, and all output neurons are also all in one substrate. Because of this the calculations can be simplified a bit. Firstly all parts of the network are affected by the one input

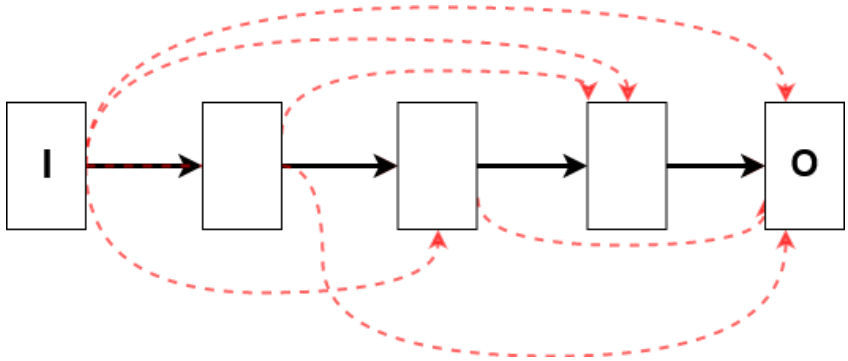


Figure 3.8: Lowest possible connection length ratio results in one information path

substrate, and the output substrate have been affected by every substrate in the network. Also any substrates in the first hidden layer can only receive signal from the input substrate, and thus it will always be connected to one out of one possible. Because of this the input layer, first hidden layer, and output layer don't hold much useful information in regards to interference, and are not calculated into the average score.

Connection Length & Interference

Using both Connection Length Cost(CLC) and the Interference score together have some advantages and they compliment each other well. They both gives maximum cost when the network is fully connected. Interference will also give maximum cost when the network never branches, something that makes up for CLC's shortcoming of favouring too sparse networks. The CLC is anticipated to encourage that the multiple pathways encouraged by the Interference cost consist of many short connections between close neurons. Using both should encourage modular networks to form, and the total Connection Cost(CC) is calculated as the average of CLC and the Interference cost.

Since both the CLC and Interference cost are ratios, they will both range from 0 to 1, where 0 is the most modular, and 1 the least modular. The CC should encourage modular topology, but it is important that id does not dominate the fitness evaluation to much. If the CC becomes so important that performance is rendered inconsequential, it will be hard for evolution to keep high performing individuals in the population. This is not favourable because the ANN would not optimize for performance as well. There is also a danger that F-modularity

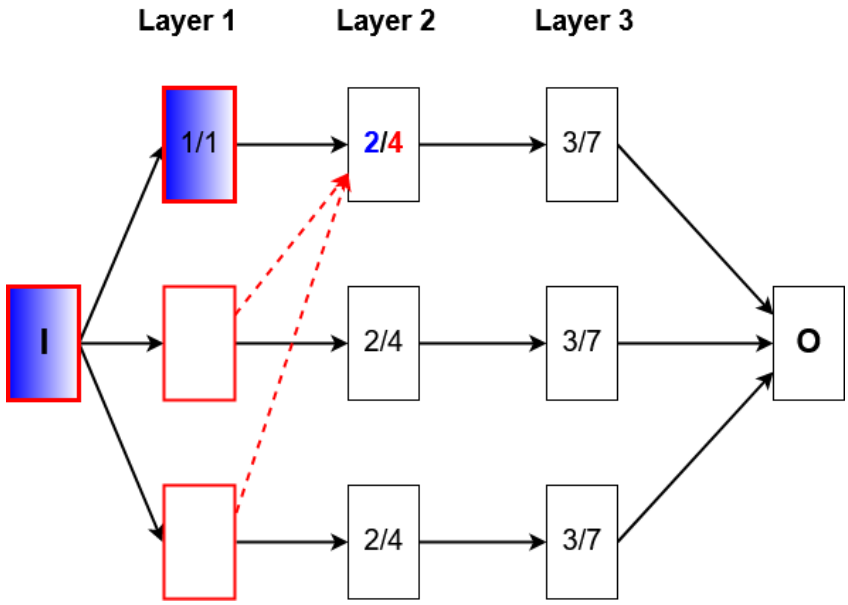


Figure 3.9: Interference is measured by how many substrates affect a substrate, and how many it potentially could influence it.

is never achieved. If for example a solution require F-modularity, then performance on a task should not be neglected for the sake of forcing a T-modularity. As discussed in section 2.1.3, having achieved a T-modular ANN does not guarantee that the ANN actually solve the problems i a modular manner. Since F-modularity is anticipated to give an improved performance, it is important that performance has an relevant impact on the fitness. For this reason the CC is not used in every generation, but instead each generation have a certain percentage change of incorporating the CC into the fitness evaluation. To not drag the population to heavily in two different directions each time CC is used, the CC is also halved, so that modularity don't overshadow performance, like it is discussed in section 4.3.2.

Discarded Alternative for Interference

Since the input substrate is always included, one alternative way could have been to not count the Input substrate when counting substrate influence, and all possible maximum number of substrates influences. Not counting the input substrate

would results in 1/3 in all substrates in hidden layer two and three instead of 2/4 and 3/7 in the Figure 3.9. In this case it would better reflect the number of pathways through the network, and keep the number constant as long as the pathways did not interact, as illustrated in Figure 3.10. But this would remove cost for solutions that connect to only input, making evolution strongly favour substrates that only connects to input and output. In practice result in only one hidden layer. This was deemed not favourable and therefore it was decided to count the input substrate when evaluating how big part of the network affect a substrate.

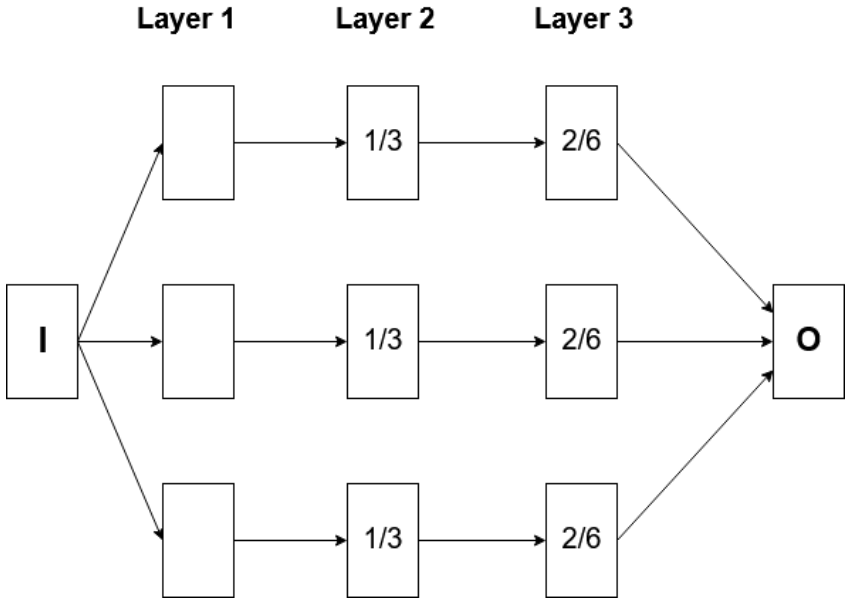


Figure 3.10: Discarded alternative where input are not counted when calculating interference

3.6 CPPN Output Structure

Earlier work that have evolved the substrate structure have decoded it from the CPPN as discussed in section 2.2.6. Since this work have a substrate structure that is independent of the CPPN, there is more room for how to use the CPPN. Notable choices are the number of inputs, what types of CPPN output to use,

and how many CPPN outputs used, as discussed previously in 2.2.5. The input to the CPPN are discussed earlier in section 3.4, so this section will focus on the CPPN outputs and three different alternatives that can be used with the EMSS-HyperNEAT. The three variants for how to use the CPPN output consist of two known methods from the literature, and one new.

3.6.1 Classical CPPN

The first one uses the one described in section 2.1.5 with the inclusion of the LEO output described in section 2.2.1. This variation will have three outputs from the CPPN. One LEO output to read off to determine the existence of connections, one CPPN output dedicated to the weight of connections between neurons, and lastly a CPPN output used when determining biases for neurons in the ANN. For simplicity these three together will be called a *CPPN output set*. This variation is appealing because it is easy to implement and does not need to change during evolution to accommodate the changing substrate. To separate this CPPN variant from the others it will be referred to the *classical CPPN variation*. Because it only has three outputs it has the lowest complexity of the three variations that will be presented.

3.6.2 MSS-CPPN

The second variant is the one described in section 2.2.5, where there exist CPPN output sets for every possible connection between substrates in the substrate structure. This is the most common solution used in the literature when the substrate structure consist of multiple substrates, and thus it might work well with EMSS-HyperNEAT as well. It also provides evolution freedom to to customize the outputs for each substrate. Because it allows evolution to specialize CPPN output patterns the most for a given substrate structure, this could be a well performing alternative. Because this variant is the most common when MSS is used, it will be referred to as the *MSS-CPPN variation*. Because the it will end up with the most CPPN outputs as discussed in 2.2.5 it is the variation that introduces the most complexity. There is a risk that this problem will be magnified by the fact that the size of the evolving substrate are not restricted and could potentially grow quite large. Another problem is that each addition of a new substrate will add to the CPPN output neurons 3*(number of existing substrates already existing in all other layers). This large increase in complexity of the CPPN might discourage evolution from increasing the size of the substrate structure, constricting the evolution of the substrate. MSS-CPPN have greater potential for adjusting the CPPN to each substrate connection, but the complexity might make it hard to fully utilize this potential since evolution have so many more outputs to optimize for compared to the Classical CPPN version.

3.6.3 1:Substrate CPPN

The third variant is an attempt at overcoming the anticipated scalability challenges of MSS discussed earlier in section 2.2.5. Instead of adding new CPPN output sets for each possible substrate to substrate connection, it adds only one CPPN output set per substrate. Figure 3.11 shows how one CPPN output set is used per substrate, giving eight output sets where the same substrate structure are shown to give 22 when MSS-CPPN were used in Figure 2.11. With this design all connections in to a substrate are decided by a dedicated CPPN output set used only for that specific substrate. This design is intended to reduce the complexity, while still allowing each substrate to have a specialized output pattern that is distinct from other substrates. It is a middle ground between the Classical version with only using only one CPPN output set for all substrates, and the MSS version that risk having too many outputs for evolution to effectively manage. Because there is one CPPN output set per substrate it is called the *1:Substrate CPPN variation*. Because an evolving substrate don't have a restriction on the number of substrates, it is probably a better fit than the MSS-CPPN variation if the evolution chooses very large substrate structures. On the other hand it does not have as specialized outputs, having one output used per substrate, instead of one for each substrate to substrate connection. It still offers more separation between different substrates than the Classic-CPPN variation, something that probably a useful tool for evolution when the CPPN and the substrate structure both evolve to synergize with each other. This CPPN variation gives more complexity than the Classic-CPPN variation, but it is a linear increase so it is still a lot less than MSS-CPPN.

These three variants have all a tradeoff between complexity and how separated and specialized each CPPN output is. With the evolutionary freedom to both grow large substrate structures, and the need to have the CPPN be able to take advantage of the substrate being evolved, what alternative that works the best is not clear. All the versions have positive and negative aspects. Classical CPPN and MSS-CPPN variants are the two extremes and the 1:Substrate holds more of a middle ground between the other two. It is unclear which solution that will work best, and for this reason it is a research question to find the appropriate CPPN variation. Therefore the EMSS-HyperNEAT algorithm is made to support these three different CPPN variants so that experiments can be carried out to investigate their different impact.

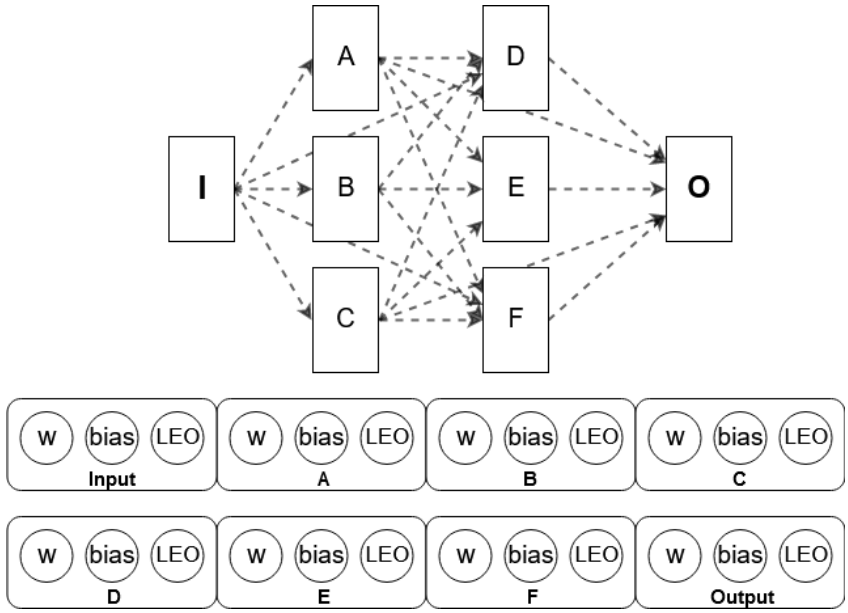


Figure 3.11: The 1:Substrate CPPN variation have one CPPN output set for each substrate

3.7 Simulator Design

The implementation is build upon the neat-python package[McIntyre et al.] and Pytorch[24], a popular machine learning framework. The neat-python package are utilized for evolutionary process that does not include fitness evaluation, while Pytorch are used to run, train, and evaluate the ANNs. The custom made modifications that needed to be build for the evolutionary process are the genome, the genes, manipulation of the genome such as mutations, and the fitness evaluation function. Pytorch were chosen because it uses *define-by-run execution*, and there is no need to set up static models like other alternatives such as Tensorflow and Caffe. Given that the structure of the ANNs changes during evolution this was seen as a better fit. The *Immediate, eager execution* feature also allowed CPU and GPU computation to be pipelined. The core logic of PyTorch are done in C++ making it run quickly, while a Python API makes it easy to develop prototypes. Another benefit with PyTorch is that the it is designed to group group neurons together for computationally efficiency. This fits well with the idea of gathering neurons into substrates. Choosing these two allowed the entire imple-

mentation to be done in Python.

The fitness function is where the majority of the processing happens. The fitness function takes in a genome, and returns a fitness that can be used by the NEAT evolutionary algorithm. How this is implemented is depicted in the Flowchart 3.7, where all the boxes are implemented in Python. The following is a brief description of each step as it happens chronological in the implementation.

(A) The fitness function begins by receiving the genome to evaluate. This genome consist of chromosomes for the CPPN node and connection genes, as well as for the different genes used by the substrate structure

(B) Disabled genes are filtered out so that they are not transcribed.

(C) In this stage two things are done:

- the CPPN genes are turned into a ANN that can run in PyTorch. The CPPN are a list of the outputs with corresponding computational graphs.
- The relevant genes from different chromosomes related to the substrate structure are grouped together, as explained in 3.2.

(D) At this stage there is a functioning CPPN, and a substrate structure without any connections.

(E) The coordinates from the substrate structure and the CPPN are used to create the connection matrix discussed in section 3.4

(F) Any invalid connectivity is handled. This includes when signals never reaches the output, if substrates don't ever transmit or receive anything. How this is handled is discussed in section 3.4. If any substrates are removed and 1:Substrate CPPN or MSS-CPPN variations are used then the corresponding outputs are also removed from the CPPN.

(G) At this stage the connection matrix has its final from and the network topology is decided.

(H) If the current generation is using CC then calculation cost can be calculated at this point with the connection matrix and the coordinates from the substrate structure.

(I) A Pytorch computational graph is created for each substrate S as a Linear module, (meaning each neurons value is decided by input*w+bias). To create this object it needs the number of input neurons and output neurons. The output neurons are simply the number of neurons in the current

substrate S, and the number of input neurons are the sum of neurons in all substrates transmitting to the substrate S. One such Linear object is created for each substrate. These "Linear objects" are the ones that holds the weight and bias values. Each substrate will have a Linear object and an activation function that will be used when the ANN is run.

(J) The Linear objects created in I were initialized with random values, so the next step is to use the CPPN to set correct initial weight and bias values. The implementation of this vary a bit depending on what CPPN variation from section 3.6 that are used. All coordinates vectors descried in section 3.4 that are used for the same receiving substrate (or for each pair of substrates in the case of MSS-CPPN version) are gathered together so that they can all be run through the CPPN at the same time. The output from the CPPN is then reshaped to be able to replace the weights in the Linear object for the substrate. This is done first for the weights and then for the bias values. After all this is done the ANN have all the correct values.

(K) The ANN is trained and evaluated as described in 3.5. For each training batch, the training data is split into input and labels, then the input is given to the ANN and the output is compared with the labels. The loss(error) is calculated and used by the optimizer (Adam) to adjust the weights and biases in the correct direction. In many ways this is how PyTorch usually is used, mainly to use BP to train one single ANN topology.

(L) If Connection cost is used then the performance evaluation from **K** and the CC from **H** are averaged to give the final fitness score. If not then the fitness score is just the performance evaluation from **K**. This is returned to the NEAT evolutionary algorithm to be the fitness of the genome, so that it can be compared against other genomes in the population during the evolutionary process.

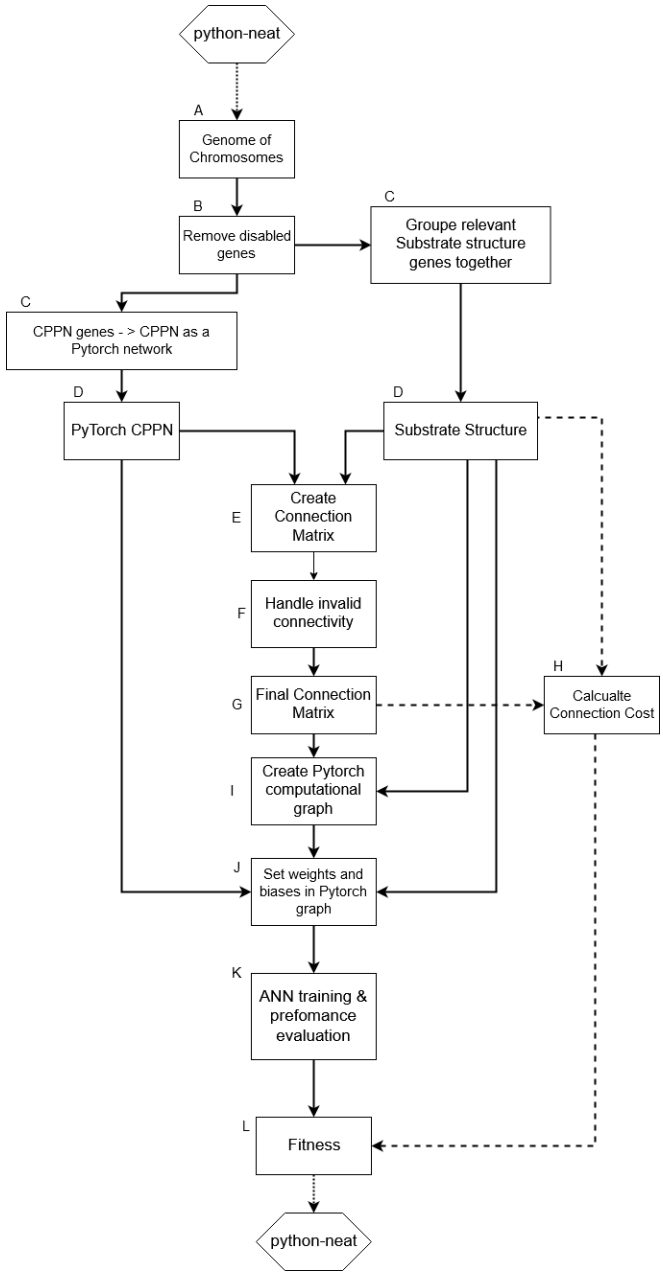


Figure 3.12: Flowchart for the fitness function

Chapter 4

Experiments, Results and Analysis

This chapter presents experiments designed to answer the goal and research questions laid out in Section 1.2. The experiments are set up to illuminate what benefits we get from an evolving substrate structure, and if it can achieve modular ANNs.

The chapter starts with presenting preliminary testing done during development in section 4.1. These informed the choice of parameters that are used in the experiments and these are presented in the next section 4.2. After this the following sections presents four sets of experiments, each aims at answering a RQ. The first two set of experiments are designed to find the best CPPN adaption. The best one will be used in the subsequent experiment, which tests the model on more modular problems. The last set of experiments are designed to compare the model against HyperNEAT where the substrate don't evolve. For each experiment, the method for the experiment are presented, followed by the results of the experiments and an analysis of the given results.

4.1 Preliminary testing

To not bias the parameters towards modularity, the preliminary experiments were first conducted on the MNIST data set. The MNIST data set is a collection of images hand written of digits with corresponding labels, and is a common benchmark often used in image classification. Modularity is not required when classifying MNIST images, so this was chosen to avoid fine tuning the parameters specifically for modularity. This is to increase the odds that the results

gathered from experiments on modular problems are a consequence of the EMSS-HyperNEAT design, and not a consequence of specifically chosen parameters for those problems.

An important parameter to set here are the learning rate, since a higher learning rate would change the weights faster, but also make for a more volatile fitness curve throughout training. This could potentially make it a bit random if the training stops at a point of high or low accuracy. During evolution it is important to have an good way of comparing the quality of two individuals, so this randomness was a problem. If one individual that on average perform quite poorly, but get lucky and happens to by chance perform well in the comparison, can out compete an individual that on average performs better. This could mistakenly add individuals with the property of being very volatile and inconsistent in regards to how they improve with BP training. This problem of consistent performance score for an individual were also discussed in section 3.5. Because of this there exist a certain trade-off between requiring fewer training batches because weights change more each batch, and the consistency of the performance score give to one individual. To test what learning rate should be used, the same ANN were evaluated with learning rates 0.004, 0.002 and 0.001.

The ANN were created by doing an evolutionary run using 0.004 learning rate for 100 generations, where performance were measured as classification accuracy at the end of training period. The resulting best individual produced by evolution were then used to test how training with the different learning rates would affect the consistency of the classification accuracy achieved on the same ANN. Each learning rate was trained and then evaluated ten times.

Learning rate	mean	std
0.001	0,8392	0,0103
0.002	0,8012	0,0203
0.004	0,6725	0,0403

Table 4.1: Preliminary testing: Variations of achieved performance of the same ANN trained with different learning rates

For different learning rates the same ANN achieved distinctly different accuracy and also different variation. The results are shown in Table 4.1. A higher learning rate used during training resulted in an increase of the variation of the final classification accuracy achieved after training. The evaluation of the best individual achieved through evolution varied twice as much more with 0.004

learning rate than it did with a learning rate of 0.002. Learning rate of 0.002 also had twice as high as 0.001. Higher learning rate allows for the ANN weight and biases to change more per training batch, and this seem to prevent the ANN from consistently stabilizing on one good solution. Based on this the learning rate was lowered down from 0.004, and a learning rate of 0.001 was used during the following experiments.

The preliminary tests on MNIST also gave an indication on how many generations the algorithm used to converge when it used training. When training is used each generation takes longer time, and because multiple evolutionary runs would be used for each experiment, it was useful to find the point at which improvements was strongly diminished, so that time and computational resources could be spend the most effectively. On figure 4.1 there is a clear indication that the population converged around 0.8 fitness in relatively few generations. The results for MNIST usually converged around generation 20, and this became the number of generations used when BP training was applied in the experiments. Because the MNIST is a harder problem than the modular problems, it is assumed that they should not take any longer time to solve.

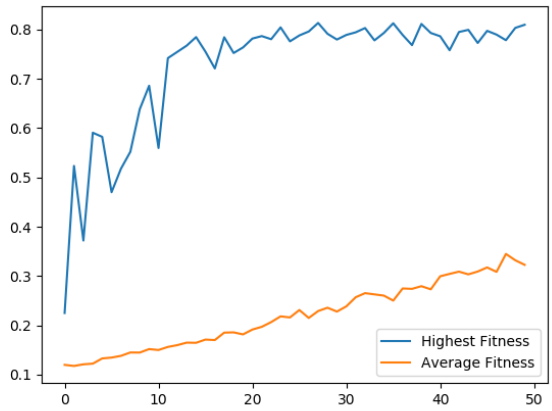


Figure 4.1: Fitness history for a preliminary test on MNIST.

4.2 Parameters

Many of the parameters did not appear to have large impacts on MNIST during the preliminary testing, and would usually always achieved very similar results for different parameter choices. For this reasons, the rest of parameters choices were instead based on similar work in the literature. The literature chosen can be seen in Table 4.2, and are chosen from work on HyperNEAT that have aspects that are similar to this EMSS-HyperNEAT. The two most left columns are the parameters chosen for EMSS-HyperNEAT with and whiteout training. Right of this is CCT-HyperNEAT[13] which also uses Connection Cost and run many of the same problems that this project will use in the experiments. CCT-HyperNEAT were further discussed in section 2.2.1. To the right of CCT are DeepHyperNEAT[31] and ES-HyperNEAT[26] which both in some way altered the substrate structure during the evolutionary run, as discussed in section 2.2.6. In DeepHyperNEAT[31] the substrate structures are decoded from the CPPN, and ES-HyperNEAT[26] chose what part of the substrate to use based on the CPPN output pattern. Right of ES-HyperNEAT is the column HyperNEAT+BP where HyperNEAT was combined with back propagation learning[35]. All of these does something that are similar to EMSS-HyperNEAT, either encouraging modularity, changing the substrate, or combining HyperNEAT with BP learning

Parameter	With BP	No BP	CCT	DeepHyperNEAT	ES-HyperNEAT	HyperNEAT+BP
Add CPPN Node	0.18	0.1	0.05	0.2	0.02	
Delete CPPN Node	0.18	0.04	0.04	0.2		
Add CPPN Connection	0.2	0.1	0.09	0.5	0.03	
Delete CPPN Connection	0.2	0.1	0.08	0.5		
CPPN weight Mutate Rate	0.2	0.2			0.94	
Add neuron	0.2	0.1	NA	NA	NA	NA
Delete neuron	0.05	0.04	NA	NA	NA	NA
Add substrate/layer	0.1	0.08	NA	0.1	NA	NA
Delete substrate/layer	0.05	0.05	NA	0	NA	NA
Enable/disable genes	0.15	0.05				
Nodes in new substrates	10	2		1 and 3	NA	NA
Generations	20	400	2500-50000	100	1000	2500
Population size	100	200	1000	150	300	256
Training batches	200*4	NA	NA	NA	NA	250*
Batch size	64	NA	NA	NA		300*

Table 4.2: Parameters used for EMSS-HyperNEAT and comparable papers, NA are filled in if the parameter was not relevant, and blank are for values not reported. *reported to train 250 epochs on 300 images

CPPN Parameters

A lot of the other work in Table 4.2 work have run for longer generation than were feasible in this project, as can be seen in the row for "Population size" and "Generations". The CCT-HyperNEAT is the one that ran for the most generations and had the largest population size. That is the project that had the longest time for the mutations to accumulate. DeepHyperNEAT had more comparable computational resources to this project and used higher mutation rates. The CPPN mutation rates chosen for the experiments are therefore in the range between the rates reported for these two papers, and can be seen in the first five rows in Table 4.2.

Substrate Structure Parameters

The only earlier work that could increase the substrate size by the use of mutations was the DeepHyperNEAT, and as can be seen in Table 4.2 it has a 10% chance of adding width or dept of the substrate structure. It also allowed for either 1 or 3 neurons in each new substrate depending on the configuration. Because of this two was chosen as the number of neurons that new substrates are created with when training is not used. The number of nodes in new substrates are chosen to be higher when EMSS are used with training, since BP makes it easier for the ANN to adjust to the new nodes. The assumption is that additions of new nodes and substrates are less disruptive with BP. Without training it is assumed that adding too many nodes at the same time would be too disruptive. Similarly, the addition of new neurons are less disruptive than adding substrates, so the probability to add new neurons are set higher than the probability of adding new substrates and layers. This is also anticipated to give evolution some time to adjust the existing substrates before new disruptive substrates are added.

Generations and Population size

The runs with training had fewer generations because each generation takes more time, and preliminary testing seem to show that it converges in relatively few generations. Because fewer generations are used with training, it also has higher mutation rates, since it does not have as much time to let many small mutations accumulate. These differences can be seen in Table 4.2 for mutations rates for both CPPN and the substrate structure. Because the time spend per individual is a lot shorter without training the population size is 100 with training and 200 whiteout training.

Training Parameters

For the parameters relevant for training the most relevant work for HyperNEAT+BP does not report the number of training batches and the batch size used, but instead the number of epochs run over 300 images. Seems to indicate either all 300 images at once or batch size of one. 300 batch size could not have been used in the following experiment because it would have been larger than some of the modular problems with less training data available. Large training batches also degrade the generalization of trained ANNs[17]. A single training example would have been a very small sample to approximate the correct gradient. A batch size of 64 was chosen because it would give a decent sample size and not be larger than any of the data sets used for experiments. It is also very commonly used batch size for training[17]. When it comes to the number of training batches used in HyperNEAT+BP, there is an uncertainty because the size of the training batches is not known. However, 250 epochs of 300 images means that a total of 75000 images was used during training. For the following experiments on EMSS-HyperNEAT, the parameter chosen for the number of training batches used is 1200, which means that there will be 200 training batches between each of the performance evaluation discussed in section 3.5. This gives a total of 1200*64 = 51200 examples seen during training. This is over 20 000 less than what was reported to be used in HyperNEAT+BP, but constraints on time and resources meant that this project could not go much higher. Thus, the parameters chosen for training was 200 training batches between each evaluation, batch size of 64, and learning rate of 0.001.

The size of the evolved ANNs may be affected by the resources used for training. Since smaller ANN would have less parameters to adjust, it would be faster to train up to the maximum for that ANN. This could give skewed results during evolution if better solutions simply need more time to train. In this work it was desirable that the networks trained fast, and this something that was encouraged. Smaller networks are also preferred over larger complex ANNs if they both can solve the problem. The problem only occurs if a certain size of ANN are required, but that size also are discouraged by evolution because it can't compete with the smaller, faster trained ANNs. Since the selected parameters for training are selected to be similar to what is needed for MNIST, it should be enough resources to not artificially constrict the size of ANNs evolved for the modular tasks since they are not as complex and are thus anticipated require smaller ANN.

CCT Parameters

CC was used in the fitness evaluation in 25% of the generations, similarly to how frequent it was used in earlier work that simulated physical constraints for modularity in ANNs[13; 7]. As discussed earlier in section 3.5.1, this is to have stronger selection on ANN performance than on connection costs.

4.3 T1: CPPN Adaption for Modularity (RQ1)

The first set of experiments are aimed at answering RQ1 "*How should the CPPN be adapted to achieve modular neural networks?*". Towards this goal the experiments consist of trying three different alternatives, and see if they result in modular solutions.

As discussed in 2.1.3, a T-modularity in an ANN does not mean that the calculations inside the network are done in separate modules. Because the existence of this F-modularity can't be confirmed by measuring the T-modularity it is challenging to know if the ANN actually solve sub-problems inside discrete separated parts of the network. To investigate if the CPPN adaptions can achieve ANNs that processes F-modularity the EMSS-HyperNEAT algorithm is tested on a problem that consist of the separate sub problems where interference between unrelated sub problems are detrimental.

4.3.1 T1: Experiment Setup

To see if the CPPN adaption can achieve F-modularity, all the modular problem from section 2.2.2 were considered. The retina problem have the a clear left right symmetry in the inputs, but for modularity it is the simplest one since it only has two modules. The 5XOR problem consist of five fully independent sub functions, and have the largest amounts of inputs that evolution needs to organize into separate modules. These attributes makes it an attractive candidate to compare the CPPN versions in regards to modularity. The downside of the 5XOR problem however is that earlier work have spent twice as long solving 5XOR as the HXOR problem[13]. The HXOR problem does not have as many inputs as 5XOR, but it has other advantages. Where the XOR modules of 5XOR don't interact, in HXOR some of the modules have to work together. HXOR is split into two modules same as retina, but each of these two modules contain three XOR problems where the first two are input to the third. Because of this it should be a benefit if an algorithm have the ability to develop ANNs where the modules can interact, and can divide modules further up into smaller modules. Because of these properties of the HXOR problem and because it is anticipated to need shorter amount of time to solve than 5XOR, the HXOR was selected as

the problem to compare the different versions of CPPN described in section 3.6.

The experiment plan is shown in Table 4.3.1. As described in the test plan overview the CPPN versions are tested both with and without training in case the different methods would react differently. When evaluating the best individual from each run the maximum achieved accuracy of 30 evaluations are used. To make it fair the best individual evolved without training during evolution are also trained before the results are compared. These best individuals from each run are trained on the full training set 50 times. CCT on the run without training was not halved as mentioned in section 3.5.1. To see if the different adaption methods would react different with and without training, five evolutionary runs was done for each CPPN adaption with training, and five without. The results of these experiments will determine what approach will be used in the subsequent experiments.

RQ1	How should the CPPN be adapted to achieve modular neural networks?
Test plan overveiw	Testing if modular ANN are achieved with different methods of applying the CPPN. (The CPPN output nodes for weight, bias, and LEO values are abbreviated to *CPPN output set*) • **T1** Run 5 runs of the HXOR problem on different CPPN adaptions. With and without training • **Measure:** Average and maximum performance of the best individuals, and the standard deviation of these. • **T1.1** Using a single CPPN output set (classical HyperNEAT) • **T1.2** Using one CPPN output set per possible substrate to substrate connection (MSS) • **T1.3** Using one CPPN output set per substrate (1:substrate) • **Focus of analysis:** What adaption works best for solving the modular problems, and the spatial information evolved in the substrates
Hypothesis	**MSS adaption** should perform better than **classic adaption** on small networks given that it starts off with fewer assumptions about neuron correlations, and these can instead be learned through evolution. However, it is believed that **MSS-adaption** should start to have trouble once the number of substrate grows above a certain size because of CPPN output scalability discussed in 2.2.5. Since it is hypothesised that the substrate will grow during evolution to iteratively capture more complexity, **MSS-adaption** will improve slower if the tasks require large networks. It is hypothesized that the **1:substrate-adaption** should perform similar to **MSS-adaption** on smaller networks, and perform better on larger networks because it scales better with regards to the number of substrates. Thus, hypothesis is that **T1.3** should perform the best overall. The input neurons are expected to group together such that those that would go into the same XOR function would be located close together, so that it would be easy for the CPPN to connect them to the same parts of the ANN

4.3.2 T1: Results and Analysis

T1: F-Modularity Without BP Training

The results of experiment 1 are shown in Table 4.3.2, and there were no large differences in between the different adaptions. On average the MSS adaption achieved ANNs with 42.34% accuracy, slightly more than 1:substrate and classical adaption who on average achieved 41.89% and 39.69%.

The maximum accuracy achieved shows a bit more difference, with 1:substrate achieved a bit better maximum accuracy(53.1%), while the classical and MSS adaption achieved 51.1% and 48% during the experiment. The reason why classical adaption have the lowest average but not the worst maximum accuracy is because it varies a lot more than the other two approaches. Amongst the different runs for the adaptions, classical had a std. of 0.105 , while MSS and 1:substrate being more a bit more stable with std. of 0.042 and 0.066. Thus, the classical variation can achieve better accuracy than MSS, but does so a lot more infrequent. Overall it seems like the EMSS-HyperNEAT did not solve this task within the 400 generations allowed in this experiment. A possible explanation for why it did not perform well is that there were not a strong push away from spatial interference without the use of any BP training. This was a one of the reasons why EMSS-HyperNEAT are designed to be able to use BP learning as discussed in section 3.5, since F-modularity benefits from a learning algorithm that promotes specialization of the modules within the ANN. Other work have used 25000 generations and population size of 1000, so this might also have been a matter of simply not giving it enough resources. As discussed in section 2.2.1, earlier work with CCT have had a tendency of optimizing for modularity in the later part of the evolutionary run, so it could be that 400 generations were not enough for CCT have a strong enough impact on modularity.

CPPN adaption	Mean	std	Max Acc
1:Substrate	0.41875	0.0656	0.53125
Classic	0.3969	0.10577	0.5117
MSS	0.4234	0.0444	0.4804

Table 4.3: T1: Accuracy results for CPPN versions on HXOR without training

T1: Effects of CCT

The effect of CCT are very visible in the evolutionary run, since they produce distinctly lower fitness values when the CC are subtracted from the accuracy score. This is the reason for the many dips that can be seen in Figure 4.3.2. The 75% of generations without CCT only tries to increase the accuracy of predictions, while CCT tries to lower the CC.

This usually result in a periods where the GA increases accuracy and the connection cost. Then CC are turned on and the GA tries to minimize the connection cost, and possibly sacrificing some accuracy progress made in order to achieve it. In Figure 4.3.2 the black arrows shows how individuals that achieve higher performance seem to disappear from the population. CCT also introduces a lot of variation since the population is frequently pulled heavily in two different directions. Generations where CCT were applied often saw bigger sparser network achieve the highest fitness, while regular generations were slower to change the substrate structure itself. Thus, CCT seemed be a source of more risky variation of the substrate structure, often favouring large sparse networks.

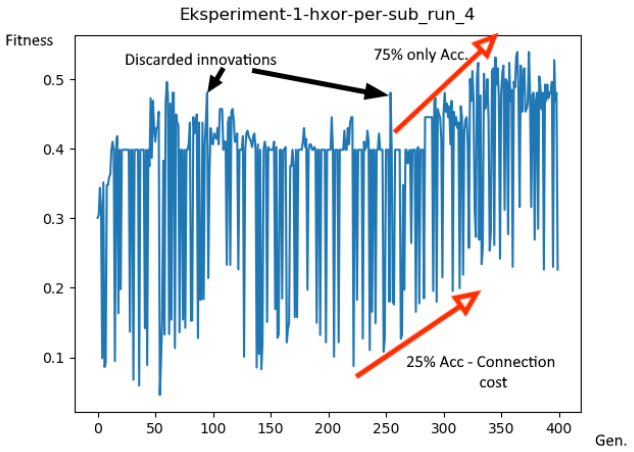


Figure 4.2: Graph showing the effects of CCT on fitness throughout evolution for the HXOR problem without training

The heavy disturbance of the CCT might be because the CC importance are considered as important as performance when it is used. Since CC span from 0 to 1, same as accuracy, the connection cost would often be of a relatively similar

size as the accuracy. If for example a new innovation resulted in a 5% increase in the accuracy as can be seen with the black arrows in Figure 4.3.2, and if the CC varied by a lot more than this, then there is a risk that the differences in accuracy performance are so small that they accuracy performance are not relevant for fitness comparisons. This were discussed earlier in section 3.5.1. This could risk the CCT generations in reality only optimizing for lowering CC. This would explain that accuracy sometimes take a big hit after a generation of CCT, and especially if multiple happened sequentially.

This could be unfortunate if recent promising innovations have not had time to develop, see black arrows in Figure 4.3.2. The design of the CCT is to discourage developing fully connected solutions with low chance of modularity, but it seems like it was a bit to strict. Innovation was discarded before they had a chance of diversifying into more modular alternatives. As a response to this, the connection cost in multiplied by 0.5 in the following experiments to lessen the disruption, and not having connection cost dominate the fitness evaluation to much. Performance of the individual should not be inconsequential in the generations that uses CCT, but it should rather give a benefit to individuals that perform well, and have low CC.

The CCT provide a lot of disturbance in the population. This is kind of like introducing noise, but also sort of similar to evolving for varying goals, given that 25% of generations have an additional goal to lowering Connection Cost. Since one theory for the benefit of modularity in evolution is variational adaption as discussed in 2.1.3, this could have had an impact on the modularity, but not enough to solve the HXOR.

The times where CCT was taking into the fitness function, larger networks tended to be the best in the generation. It seems like CCT are good for introducing variation, and has a bias towards networks with higher "potential maximum connectivity length". Thus, it was a driver for both variation and larger networks where the new addition are not fully connected to the old network. The networks also did not continuously grow throughout evolution, suggesting that the design of the new CCT design are successfully in not forcing substrate structures in the population to grow for no reason.

T1: F-Modularity with BP Training

When EMSS-HyperNEAT was used with training, significant improvement were shown for all CPPN adaptions. While all the adaptions had problems without training, with training all the CPPN adaptions are able to reach 100% performance, shown in Table 4.4. This makes it a bit hard to draw conclusion from the

experiment on what CPPN adaption are the best, because the HXOR test seem to be to hard without any training, and to easy with training. The variability for the classical CPPN adaption seen without training disappeared since it consistently were able to solve the problem. 1:Substrate and MSS also performed very well, but not reaching 100% in all runs left them with slightly lower average accuracies of 98.7% and 97.9%. It seems like all of the adaptions work well, but the classical HyperNEAT version seem to perform slightly better.

CPPN adaption	Mean	std.	Max Acc
1:Substrate	0.9867	0.029697	1.0
Classic	1.0	0.0	1.0
MSS	0.97890625	0.03045	1.0

Table 4.4: T1: Accuracy for CPPN versions on HXOR with training

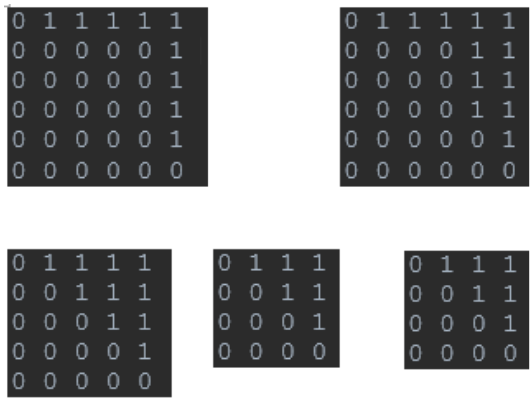


Figure 4.3: Connectivity matrices for all 5 runs of HXOR using Classic adaption. All achieved 100% accuracy, despite some being fully connected

Interestingly some solutions for the HXOR problem evolved with training were fully connected. All 5 runs for Classic achieved solutions that had 100% accuracy, yet the connectivity matrices in Figure 4.3 show that some of them were fully connected. The matrices that depicts a fully connected substrate structure is in the second row, and can be identified easily by having the upper right part of the matrix filled with ones. Thus, EMSS-HyperNEAT has at times solved the F-modular problem whiteout a T-modular ANN. For the fully connected solutions,

probably one of two things have happened:

- In the XOR problems, any input from another XOR input would not be useful, and perhaps also just disturbing. It could be that the BP have adjusted the weights for the connections between neurons so that this interfering "noise" is minimized. This would mean that the BP have separated the neurons inside of the substrates into modules, and thus separating the substrates were not necessary. This could have been aided by the fact that spatial interference would be very damaging for an ANN trying to solve this problem, and this gave an evolutionary advantage to ANNs that had F-modularity. If modules were organized within the substrates and the neurons in the same substrates calculate separate xor functions it would not matter much if the substrates were fully connected together. This type of modularity would not be visible in the substrate to substrate connections. This could explain why the substrates were fully connected, and the ANN still found a perfect solution.

- Another possible explanation is that or that the network has and somehow learned all possible input to output combinations without solving it by the expected way of calculating the xor functions in order. This is a potential problem with the small size of the problems as discussed in section 2.2.2. This could be as sign that the algorithm to easily allows for non-modular solutions, but it would not explain why there were differences between the CPPN adaptions.

Analysing the Evolved Substrate Structures

The substrates that were evolved when solving the HXOR problem shows clear differences from typical human designs. In typical substrate designs made by humans for this kinds of problem, the neurons would have been placed with regular spacing between them and the inputs to the same XOR function would be located geometrical close, shown in Figure 4.4. The evolved substrates did not show even spacing between the neurons, and input neurons to the same XOR problem were often not amongst the closest neighbouring neurons, shown in the input substrate to the left in Figure 4.5. The figure shows a substrate structures consisting of three substrates that was evolved for the HXOR problem without training. The neurons is depicted as blue dots in the substrate according to their x and y coordinates. The left most substrate is the input substrate and the substrate furthest to the right is the output substrate. Over each substrate is the coordinates of the substrate. For example the input substrate are located in the first layer and are the first substrate in that layer, which gives it the coordinates (L0s0). The coordinates are 0 indexed, so the substrate in the hidden layer have

coordinates L1s0 and output have L2s0. Each neuron in the input and output layer are depicted with an id number, so that it is possible to see if input neurons are placed according to what modules they are a part of. This numbering matches the input order from left to right shown in the problem depiction of HXOR to the left in Figure 4.4. Thus, input neuron 0,1 goes into the left most XOR and neurons 6 and 7 goes into the XOR the farthest to the right.

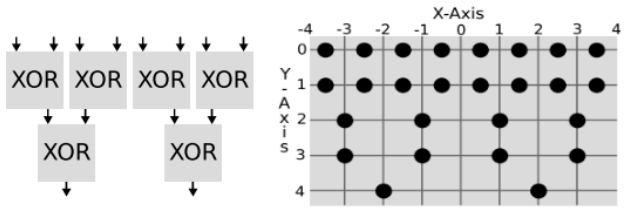


Figure 4.4: Handcrafted substrate made to mirror the HXOR problem

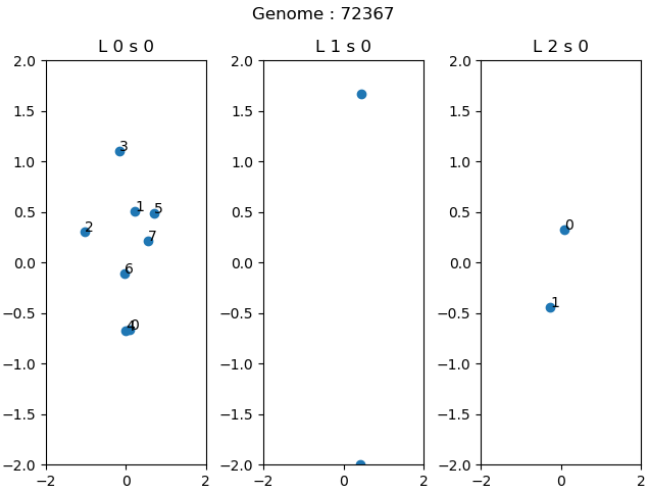


Figure 4.5: Substrate geometry evolved for HXOR problem (without training)

When EMSS-HyperNEAT was used without training it generally was a lot fonder of adding layers and substrates rather than adding neurons, even though the probability for mutations that add neurons were slightly higher. This could result in substrate structures shown in 4.6, where layers with multiple substrates

have them depicted in the same column to show that they all share the layer. All substrates have the same dimensions, but are compressed in order to show them all in one image. The reason why there seem to be a preference for many substrates with few neurons is unclear. If time would have allowed it, experiments could have been conducted to further investigate the reason for this. Experiments with varying number of neurons placed in new substrates could have investigated if two was simply an optimal number per substrate, or if it was the addition of new ones that were to disruptive. If the substrate structures often would have had the same amount of neurons as they started with, it would suggest that it is the addition of neurons are too disruptive. If it showed a preference for few neurons per substrate no matter how many neurons a substrate started with, it would suggest that evolution prefer to have more detailed control of the connectivity of each neuron.

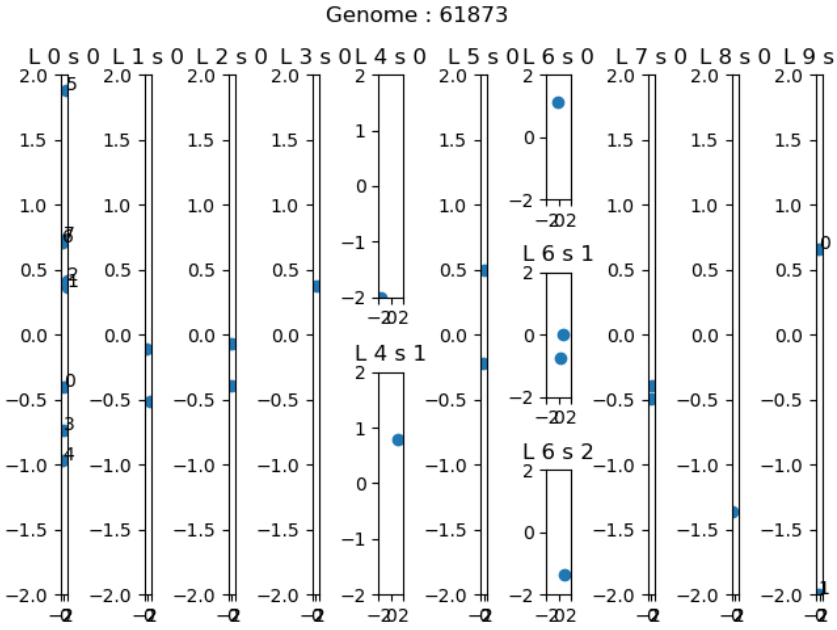


Figure 4.6: Larger substrate geometry evolved for HXOR problem (with training)

When training was used the substrates started with more neurons than they did without training. With the substrates having more than just two or three neurons, it is easier to see if the neurons in the hidden substrates form any patterns. The input neurons are not grouped together so as to be close to the other

input neuron that are input to the same XOR. Thus, the hypothesis of how the substrate geometry would from was wrong. It is difficult to know if the geometries formed were just random placement that happened to work well with the CPPN that happened to evolve, or if there is some inherent benefit of placing the neurons like they are done in the input substrate. For example the input substrate L0s0 shown in Figure 4.7 seem to have the input for most XORs in the HXOR problem separated on the y axis with one very positive and one very negative. If more time was available it would have been interesting to investigate if there are a specific substrate geometry that works well with a given CPPN, or if the placement of the neurons are placed coincidental.

Thus these results can't say that there are no geometric shapes forming, but if it did it was not the expected one. This could suggest that it can be hard for humans gain insight any into spatial aspect of problems by studying the substrates evolved through evolution in this model.

One possible reason why the inputs neurons did not group together as anticipated can be a result of all the inputs sharing the same substrate. This was done for simplicity and ease of implementation, however this means that each substrate that receive from the input layer will be forced to receive all input neurons, and are not allowed to receive only a subset. As a consequence of this a module could not receive only the relevant inputs for a XOR function, but were forced to receive all of them. Thus, each module that wanted to handle a single sub-problem had find a way of sorting out the relevant inputs in addition to evolve a way to solve it. This can have made it difficult to evolve a system that sends separate inputs to different functional modules, and could have been why evolution did not find it useful to group input neurons, since all input neurons would be transmitted together anyway. This would explain why the results were so much better with BP, since it would help the modules adjust the weights so that the relevant inputs for the module had the most impact, and limit the impact from input neurons that were not relevant.

The same problem exist in the output substrate, where any module that wants to send signal to output nodes are forced to send to all output nodes in the output substrate. It is anticipated that it would be beneficial for F-modularity if evolution could play around with dividing the input and output nodes into multiple substrates. This was however not possible within the time-frame of the project.

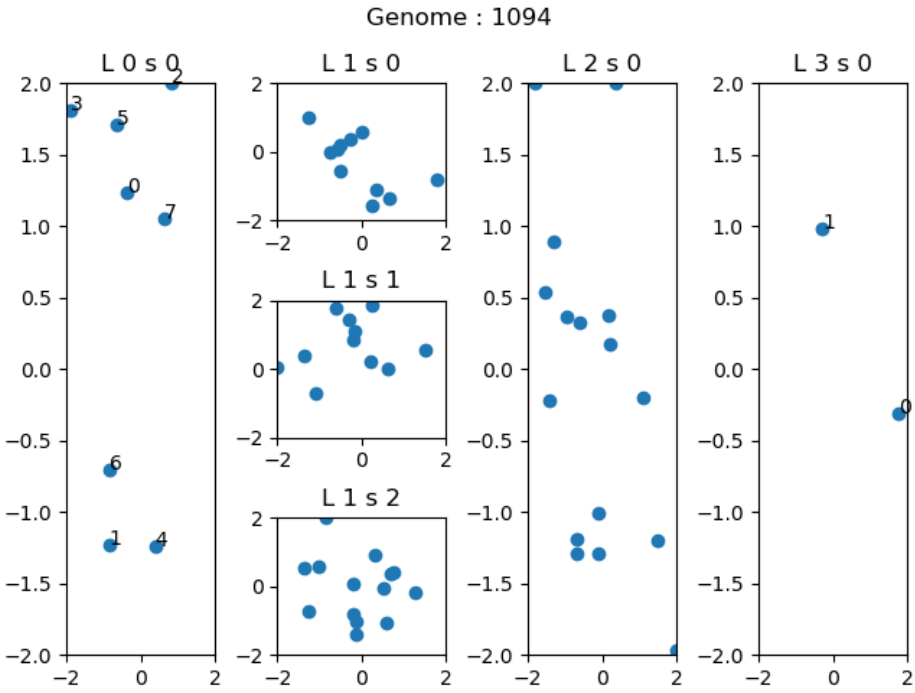


Figure 4.7: Substrate geometry evolved for HXOR problem (with training)

4.4 T2: CPPN adaption for Image Classification (RQ1.2)

The following experiment tasks in this section is about answering RQ1.2 *How should the CPPN outputs be adapted to achieve the highest performance on image classification accuracy?* described in section 1.2 These tasks measure the performance of the different CPPN versions described in 3.6 against each other and the performance reported by Verbancsics and Harguess [35], which were discussed in section 2.2.4.

Image classification allows us to see if EMSS-HyperNEAT algorithm work well when it has to solve larger problems with larger data sets and more inputs than the comparatively smaller modular problems used in section 4.3 and 4.5. Modularity assumed to generally be a useful trait, and thus it is interesting to see if the method that performs well on modularity, also are the one that perform better on an image classifying problem that don't require modularity. The EMSS-

HyperNEAT algorithm is designed for modular ANN, so if it can solve problems that don't specifically dependent on modularity would show that it is able to generalize to a wider array of problems.

The larger data sets for image classification problems means that there are more examples for each label, then was seen in the modular problems. This makes it easier to divide the data set into training set, validation set and test set. By testing the solution on a test set not used during evolution, we can know that the algorithm have found a general solution and not simply an input-output mapping for the specific data used during evolution, something that is a risk when solving the modular problems as discussed in section 2.2.2. Because results from T1 in section 4.3.2 shows that this might have happened, it is interesting to investigate. Thus, testing on image classification allows us to see if EMSS-HyperNEAT creates ANNs that generalizes to examples never seen during evolution. In summary this experiment are useful to test how CPPN adaptions scale to larger problems, see results when direct input-output mappings are not possible, and the ability of the different CPPN adaptions to generalize to a different type of problem.

4.4.1 T2: Experimental Setup

The image classification task chosen is to classify hand written digits in the MNIST dataset. MNIST is chosen because it is a popular benchmark data set for image classification, and the hand written digits have an inherit geometry where certain parts of the picture are more important, something the evolving substrates might be able to exploit.

Out of the 70 000 cases in the MNIST dataset, 54000 was used as a training set for BP training, 6000 was used as a validation set to evaluate individuals during evolution. The test set consisted of 10 000 cases. As in earlier experiment five evolutionary runs was conducted on the different alternatives to CPPN adaption. The experiment was limited to a population size of 50 due to time constraints.

RQ1.2	How should the CPPN outputs be adapted to achieve the highest performance on image classification accuracy?
Test plan overveiw	Testing of classification accuracy with different adaptations of CPPN outputs • **T2** Run multiple runs of the MNIST problem. • **Measure:** Average and maximum performance of the best individuals, and the standard deviation of these. • **T2.1** With the Classical CPPN adaption • **T2.2** With the MSS-CPPN adaption • **T2.3** With the 1:Substrate CPPN adaption • **Focus of analysis:** What adaption achieved highest classification accuracy, and if this correlate to which one achieved highest modularity in **T1**
Hypothesis	Since image classification is a relatively complex problem the hypothesis is that **MSS-CPPN** will struggle because the substrate will become so large that the number of CPPN outputs will be difficult to manage. **Classical CPPN** is expected to perform a bit worse than **MSS-CPPN** since it is more susceptible to unintended correlations, as discussed in section 2.2.5. **1:Substrate CPPN** is expected to perform the best because similar to **MSS-CPPN** it allows for clearer separation between neurons in the substrate, but it is also anticipated to scale better to larger networks.

4.4.2 T2: Results and Analysis

The results of T2 are shown in Table 4.5. The classical approach where one output set are used for all the substrates seem to be perform slightly better for image classification. The classical version achieved on averaged 92,1%, while the 1:substrate adaption that were hypothesized to preform the best achieved 90.1%, and the MSS achieved on average the worst with only 89.6%. This supports the hypothesis that the MSS adaption for CPPN has troubles scaling up to larger problems. The hypothesis about Classical having trouble because of unfavourable correlations seems to be wrong since Classic achieved highest mean and maximum accuracy. The reason Classic were anticipated to struggle is because MNIST

has 784 inputs to put into the same substrate, yet the problem of unfavourable correlation discussed in section 2.2.5 seems to not have occurred. This could be because it was very easy to move out of an evolutionary trap of unfavourable correlated neurons, since the neurons could just break the correlation by moving to a different position in the substrate by mutating x or y.

There is not many percentages separating the different methods of using the CPPN in regards to performance, but it is more than in T1 with HXOR. Both the results for image classification and for HXOR shows that the MSS performed the worst on average.

CPPN adaption	Mean	std.	Max Acc
1:substrate	0.90954	0.00270	0.9128
Classic	0.92088	0.00465	0.9269
MSS	0.8962	0.03517	0.9194

Table 4.5: T2: Achieved classification accuracy for CPPN adaptions on image classification

The variation of the different runs don't vary a lot for two of the CPPN adaptions. 1:Substrate is the one with the least variation with 0.0027, and classic CPPN have 0.0047. In the T1 experiment the Classic varied the least, so this could suggest that 1:Substrate is more consistent on larger problems, and the Classic adaption on problems with less input and smaller datasets.

In this experiment the MSS showed to be the CPPN adaption that varied the most, with 0.035. This is consistent with MSS also being the least stable for HXOR when training was used. There seem to be a larger gap between the standard deviation of MSS and the two other adaptions in image classification, than there were in the HXOR experiment. This can be seen by comparing the Tables 4.4 and 4.5. Something about the image classification problem seem to hurt the MMS-CPPN adaption more than it hurts the Classic and 1:Substrate adaptions. For example 1:substrate adaption have around ten times less standard deviation in T2 as it had in T1, while the standard deviation for MSS adaption actually increased a little.

The original hypothesis was that MSS-CPPN would have trouble because of the amount of CPPN outputs as explained in section 2.2.5. This can have made MSS adaption slower at evolving because it has so many CPPN outputs to optimize, and thus did not always have time to adjust them all and reach a perfect solution. Another reason can be that the added complexity per substrate discouraged the evolution to make an ANN that were large enough to be able to solve the problem. In this case it would make sense why the difference in variance amongst the adaptions is larger on MNIST than it was on HXOR. Another

possible reason why the MSS adaption have difficulties in this experiment could be that it produces ANNs that has worse generalization and thus has problems showing good accuracy on the test set. It could also be that the problems from HXOR are more visible when used on a more complex problem because of the anticipated troubles with scaling.

Another observation that was done during the run of the experiment is that the genetic difference between individuals in the population are higher for the adaptions that adds more CPPN outputs. This genetic difference between individuals are usually called *genetic distance*, and is used for speciation by the NEAT algorithm as discussed in 2.1.4. It is not that strange for the genetic distance to increase given that each CPPN neuron are represented by a gene, so more CPPN neurons means more genes to separate individuals. This resulted in a lot more species for the 1:substrate and MSS adaptions compared to the classical adaption. Larger genetic distances will result in more species, which is the way NEAT keeps diversity in the population. With many species the population will keep diversity and focus on exploration, while population with few species will focuses more on exploitation. This could contribute to why the classical adaption were able to achieve higher accuracy with the generations available for this experiment. The quick turn to exploitation with the Classical adaption can be seen on the left in Figure 4.8 which depicts how the average fitness of the population(orange) quickly moves close to the generation champion(blue). In contrast to this 1:Substrate adaption in the right figure does not exhibit the same exploitation, but instead the average fitness uses longer time to homogenize around the best species. Given the low number of generations, 1:substrate did not have time to discard enough species and therefore has a slow increasing average fitness. This effect did not stand out as strong in T1, probably because he 400 generation were enough to allow all the CPPN adaptions time to discard bad species and thereby move focus from exploration to exploitation.

Given these results future work that also uses few generations might want to take extra care so that evolution have time to focus on exploitation. This could be done by manual fine tuning the distance required between genomes to form separate species, use behaviour as distance instead of genome distance.

T2: Comparison to State of the Art

The results for image classification are interesting when seen compared to the earlier work on using HyperNEAT in combination with machine learning on the same problem. Verbancsics and Harguess [35] reported 58.4% classification accuracy. The EMSS-HyperNEAT managed to achieve 92% classification accuracy,

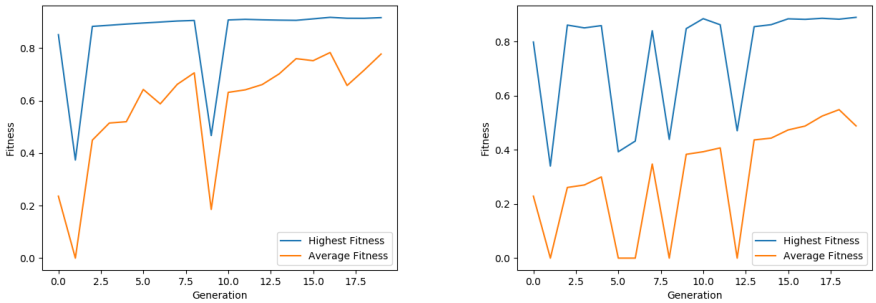


Figure 4.8: Average fitness(orange) and highest fitness(blue) in Classic adaption (left) and 1:substrate adaption (right) while evolving solutions to MNIST

suggesting that evolving the substrates can offer a significant benefit when using HyperNEAT in combination with machine learning. This seems to indicate that EMSS-HyperNEAT might also be able to generalize to problems not specifically about modularity.

A possible reason why the EMSS-HyperNEAT performed better than Verbancsics might be because of the differences in how training was done on each ANN during evolution, which were discussed in the section 4.2. The previous work trained on 75000 images seen during BP training, but many of the images would have been used multiple times since it only selected from 300 images. EMSS-HyperNEAT on the other hand were designed to train with images chosen from the entire training set of 54000 different images. So even though each ANN in experiment T2 in section 4.4 used fewer pictures during training, they probably saw many more distinct pictures and this could have been the result of the improvement.

Note that the improved performance of ANNs produced by EMSS-HyperNEAT is not caused only by BP training on a bigger selection of examples, since the reported accuracies reported by Verbancsics were the best ANNs trained with the entire training set. This means that the ANN which achieved 58.4% accuracy were had a lot more BP training than ANN that achieved 92% in the T2 experiment from section 4.4. The difference is not on how much BP were used on the topology that evolution arrived at, but rather on how BP were implemented into the evolutionary process. Thus, the number of distinct images used during evolution probably had a large impact on why higher accuracies were achieved by EMSS-HyperNEAT. Another reason for the difference can be different algorithm used for the BP optimization. Verbancsics did not report which one were used,

but it have to have been another than the Adam algorithm used in this work, since the Adam optimizer was published a year later. The design decisions on how performance is evaluated from section 3.5 might also have had an impact since they are designed to aid evolution to take advantage of the limited training resources used per generation.

4.4.3 Final Choice of CPPN Adaption

Given that the classical method of adapting the CPPN was the most effective in both Experiment 1 and 1.2 makes it the chosen adaption to use in Experiment 2. It did not work as well without BP in Experiment 1, but given how the performance improvement gained with BP, this is not seen as a big problem. The Classical variation were not hypothesised to be the best one for encouraging F-modularity with evolving substrates, so this was an unexpected result. There are multiple possible reasons for why it the classic version performed so well.

It appears to be that the added CPPN complexity of having more outputs became a greater burden than the benefits offered by the MSS-CPPN adaption and the 1:substrate CPPN adaption. One reason for this could be that the classic version were able to specialize the CPPN output for each substrate because of the design of the input vector described in 3.4. In EMSS-HyperNEAT the CPPN are given s and z coordinates in addition to the more usual x and y. It can be that this extra information to the CPPN about the neuron's position were favourable. Even if each substrate did not have its own dedicated CPPN outputs, the CPPN were given enough information that would allow it to treat different layers and different substrates within a layer differently. As a result of this the added complexity of more CPPN outputs were unnecessary. This could contribute to why the classic CPPN adaption were able to outperform the other adaptions.

The additional s and z coordinates for neurons also allows the x and y coordinates to be decoupled from the substrate's location in the substrate structure, since this is provided by the s and z coordinates. This gives EMSS-HyperNEAT two coordinates that can be altered without changing the neuron's position in the substrtate structure, but rather only changes position within its own substrate. In many handcrafted substrate like that seen in Figure 4.12, a neuron can't change the y coordinate without ending up in a different layer. The x coordinate can change without altering the layer of the neuron, but it still impacts the value of the CPPN LEO output that determines what it should be connected to. In previous work using HyperNEAT without MSS, all of a neurons coordinates impact what connections should be formed.

In contrast to standard HyperNEAT, neurons in EMSS-HyperNEAT have x

and y coordinates that don't impact the output from the LEO nodes, but only the weight values and bias values. In EMSS-HyperNEAT the x and y coordinates are used to set values for the connections, without impacting the existence of connections. This allows evolution to evolve x and y coordinates to produce good weight and bias values, and these can change and evolve without impacting existence of connections. This is similar to how LEO output are used to evolve the existence of connections separate from their values. The increased freedom for evolution could have aided it to achieve substrates structures to work well with the CPPN, even when the same CPPN output set are used for all values. In MSS-HyperNEAT the substrates connection are usually given dedicated CPPN outputs, but this work suggest that one set of CPPN outputs can be used also for a MSS as long as z and s coordinates are included.

The results from T1 and T2 could also be a result of MSS adoption and 1:substrate adoption uses longer time than Classic adoption. It could be that the other adaptions eventually would have reached the same level, or that they might even surpass the classical CPPN adaption if given enough generations, but this was not feasible to test with the resources and time available. As mentioned in 4.3.2, similar work earlier used a longer evolutionary runs than were used in this project. What is shown in the experimental results is that the classical CPPN adaption achieves the best best results within the number of generations used in this project.

4.5 T3: Functional Modularity (RQ2)

The tasks to be solved in this section expands on the experiments done for RQ1, but focuses mainly on providing an answer to RQ2: *How does ANNs produced perform on problems with geometrical relationships or clearly consists of subproblems.* In these types of problems, an ANN benefits from holding the solution to different sub problems in separate specialized modules. Because of this is is well suited to see if EMSS-HyperNEAT can achieve ANNs with functional modularity. These experiments will use the Classic CPPN adaption shown to be better in T1 and T2.

4.5.1 T3: Experimental Setup

To further investigate if the ANNs produced by EMSS-HyperNEAT can solve modular problems, the algorithm is also tested on the 5XOR problem and the retina problem discussed in 2.2.2. The 5XOR problem consist of several subtasks that need to be solved in parallel, and is therefore well suited to tests if the

algorithm can create ANNs where the connectivity between modules are limited, something that should be favoured by the CCT. To see if the *algorithm* are able to separate input into left and right symmetry, testes are also conducted on the retina problem. This task has fewer subproblems, but a clearer geometrical separation on the input.

RQ2	How does ANNs produced perform on problems with geometrical relationships or clearly consists of subproblems?
Test plan overveiw	Testing the algorithm on tasks where information from one subproblems would be irrelevant or detrimental in solving another subproblem. By solving these tasks the ANN show an ability to solve distinct sub functions separately within the network. • **T3** Test performance of the algorithm on tasks where modularity are either very useful or a necessity • **Measure:** Average and maximum performance of the best individuals, and the standard deviation of these. • **T3.1** Test performance of the algorithm on 5XOR • **T3.2** Test performance of the algorithm on retina task (L& R) • **Focus of analysis:** Performance on **T1**, **T3.1**, **T3.2**, and the spatial information evolved in the substrates
Hypothesis	The hypothesis is that the algorithm will produce modular networks, and thus perform well on these tasks. The **Retina problem** is expected to be the easier than the XOR problems, since it only needs to dividing the ANN into two modules. The retina problem also have fewer inputs and outputs than the 5XOR problem, which is anticipated to also make easier. It is hypothesised that HXOR from T1 will be a bit more difficult than 5XOR since it HXOR contains sub-tasks that are input to other sub-tasks. 5XOR makes up for some of this by having more sub-tasks that needs to be kept separate, but it is assumed that this is easier than the more complicated interactions of sub tasks in HXOR. For this reason a bit higher performance is expected on the **5XOR problem** compared to what was seen in **T1** Another hypothesis is that the inputs neurons for the left and right module in the retina problem will be geometrically divided, and input neurons to the same module will be close together. Similarly in 5XOR it is expected that the inputs that goes into the same XOR function are placed geometrically close by evolution.

4.5.2 T3: Results and Analysis

The results for T3 in Table 4.6 shows that EMSS-HyperNEAT are able to solve the modular problems. Similarly to with HXOR in T1, the algorithm consistently found perfect solutions for both the retina and the 5XOR problem.

Task	Mean	std	Max Acc
Retina	1.0	0.0	1.0
5XOR	1.0	0.0	1.0

Table 4.6: T3: Achieved accuracy on the 5XOR and Retina problem

Similarly to earlier results in T1, there were both solutions that were sparsely connected and some that were fully connected. In fact only one out of the five runs for both Retina and 5XOR were not fully connected. The model seem to be able to consistently solve tasks that are meant to test F-modularity, yet don't always develop connections that would suggest topological functionality.

EMSS-HyperNEAT is clearly able to achieve sparse networks that provides perfect solutions for the modular problems as can be seen in Figure 4.9, where the connections between numbered substrates are depicted. The different colours on the connections between substrates indicate how the information that flows through the network are kept in separate modules that don't interact before they are gathered up at the end. Even though these more modular structures were developed, the algorithm would often find it most optimal to go for a fully connected substrate structure, despite problems benefiting from modularity solutions. The reason could be that the CPPN initializes the weights such that it is easy for BP to separate nodes within the substrates as discussed in section 4.3.2. The fully connected solutions for the modular problems are intriguing, but it was not enough time to further investigate these solutions.

When it comes to how the neurons are placed in the substrates, there seem to be a bit more patterns than was seen in the HXOR problem. Four of the five substrate structures evolved for the retina problem are depicted in Figure4.10. Many of the substrates evolved to group the neurons around the middle, and these substrates had a tendency to be located in the later layers. However, grouping of neurons can also be seen in both substrates that are located first hidden layer of in the bottom right substrate structure. This grouping around the middle can be because of the CCT. Substrates where the groups of neurons are located in the same x,y coordinates will have lower connection lengths, discussed in seciton 3.5.1. If a substrate with grouped neurons are connected to a substrate where the neurons are spaced out evenly, then having the group located in the middle will

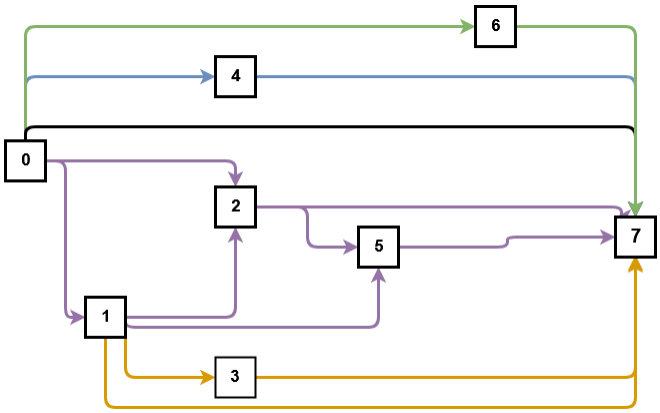


Figure 4.9: Substrate structure with T-modularity evolved for the 5XOR problem.

give the lowest connection length on average. Thus, having the neurons grouped around the middle of the substrate can be favourable to minimize connection cost.It is also possible that the grouping is a consequence of the CPPN, where the CPPN tend to give unfavourable outputs in the regions that are unoccupied.

Not all the substrates had the neurons grouped together, in many of them the neurons seem spread out. The reason for this could be that more distance between the neurons makes it easier for the CPPN to distinguish between each neuron, giving a specific output to each. It is also possible that the geometry of the neurons in the substrates without grouping are a a result of random placement. As discussed in T1, it is not always easy to have an intuitive understanding of why the substrate geometry was evolved to be what it is.

The substrates evolved for the 5XOR problem seem to be more chaotic than for the retina problem. Four of the five runs for 5XOR are shown in Figure 4.11. One thing seen in these substrates that was not seen in the retina task is that there are substrates where there is a bigger gap between two tighter groups of neurons. This can be seen in the top right image in L1 s0, and in top left image in L3 s0. This seem to happen rarely though, so it could also just have occurred by chance. Something that were also seen in the retina problem that also occurred with the solutions for the 5XOR problem is that the coordinates for the neurons seem to often be in similar regions of the substrates. When the neurons do group together it seem to often happen around the middle of the substrate, probably to minimize connection cost. In summary the substrates in a substrate structure

seem to have the a tendency to have neurons located in the same regions, forming groups that have low connection distance to groups in other substrates. This is however no strict rule, as many substrates also have a larger spread and the location for the neurons in these looks fairly random.

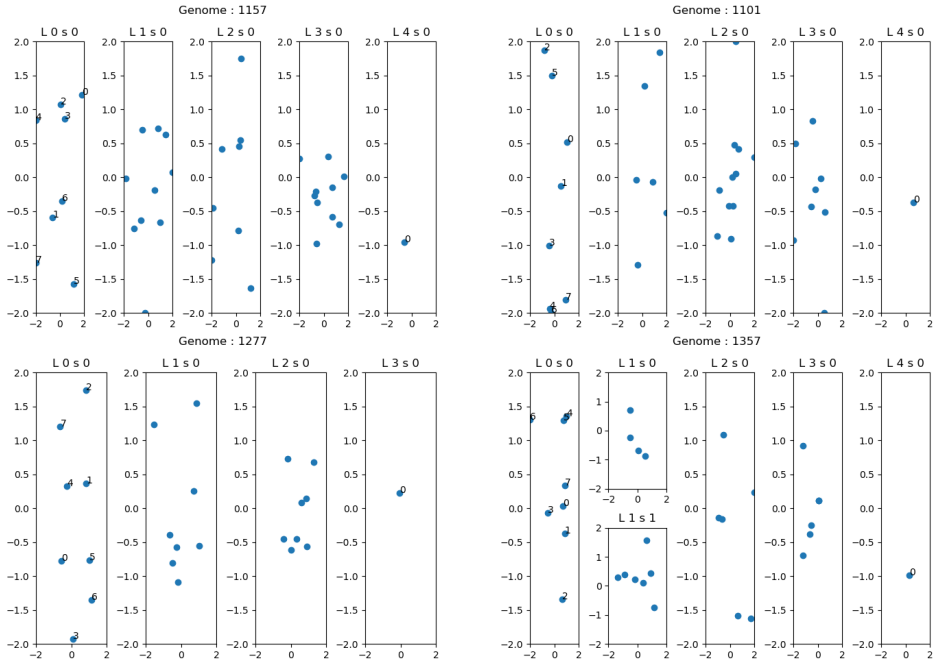


Figure 4.10: Substrate geometries evolved for the retina problem

4.6 T4: Modularity with Handcrafted Substrates (RQ3)

The experiments in this section are designed to help answer RQ3, "How does the evolved substrates perform on modular problems compared to manually designed substrates?". The experiments in section 4.3 provide the results of evolving substrates. This section will thus focus on answering how HyperNEAT will perform on a modular problem if it can't evolve the substrate, but instead uses a handcrafted substrate and only evolves the CPPN.

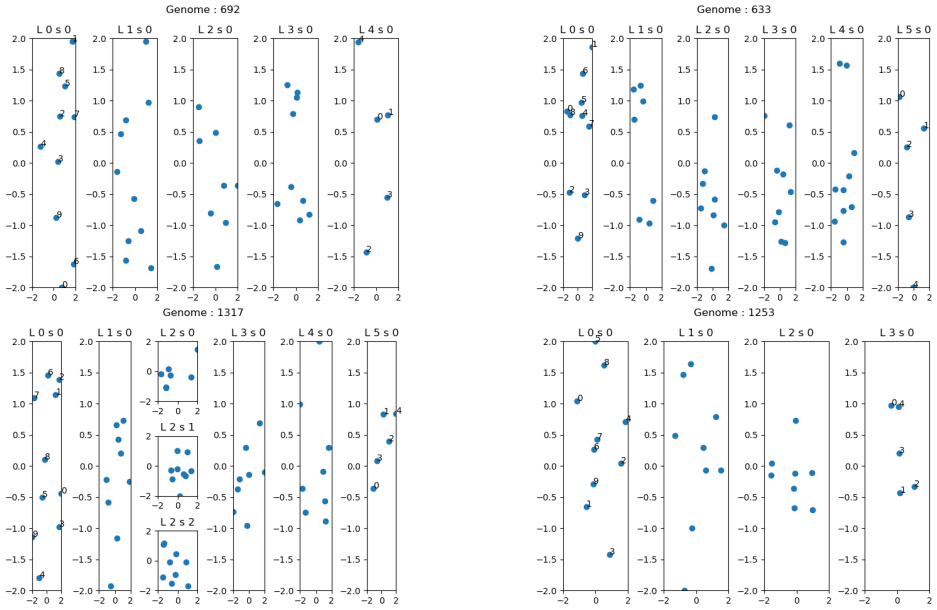


Figure 4.11: Substrate geometries evolved for the 5XOR problem

4.6.1 T4: Experimental Setup

In this experiment a hand designed substrates are applied and evolution of the substrates is turned off so that the model resembles standard HyperNEAT.

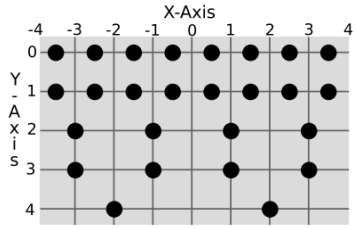
HyperNEAT connects nodes and the EMSS-HyperNEAT connects substrates, so each hidden node is placed in its own substrate. Additionally standard HyperNEAT is usually implemented with only x and y dimensions, while EMSS-HyperNEAT is designed to use two more coordinates per neuron. This is however not a problem and does not need to be changed to resemble standard HyperNEAT. This is because the two extra coordinates represent the same information as the x and y in the handcrafted substrate. The extra coordinates in EMSS-HyperNEAT consist of z and s. The z coordinate represent the layer of the neuron, and in the handcrafted substrate this is also what the y coordinate represents. Therefore y and z will be have the same value. In the handcrafted substrates, the x coordinate separate neurons in the same layer, and since each neuron is in its own substrate, this is also exactly what the s coordinate will represent. Because this there were no need to adjust EMSS-HyperNEAT to use fewer input to the CPPN to have it resemble standard HyperNEAT.

The implementation of EMSS-HyperNEAT is limited to have all input and

output gathered into one input substrate and one output substrate for simplicity. This means that each substrate that receive from the input layer will be forced to receive all input neurons, not a select subset. This limitation usually don't exist in standard HyperNEAT, where each neuron form their connections independently from all other neurons in the same layer. Because of this the input neurons in standard HyperNEAT can be connected to completely different parts of the ANN. This could not be adjusted with the available time, and is therefore kept the same as EMSS-HyperNEAT. This means that the standard HyperNEAT used in this experiment will share the constraint discussed in section 4.3.2 with EMSS-HyperNEAT. Since the standard HyperNEAT will share this constraint with EMSS-HyperNEAT, comparisons between them does not need to account for this factor.

Because the HXOR is assumed to require the more complicated modularity structure than retina and 5XOR problem, it is expected the differences are the most visible. For this reason the HXOR is chosen as the modular problem to investigate if the handcrafted substrates can solve modular problems. If more time were available it would be interesting to also test this on the 5XOR problem. 5XOR focuses more on the separation of modules, and it could be the case that this was easier or more difficult in evolved substrates compared to the ones that were handcrafted.

The hand-crafted substrate is taken from Huizinga et al. [13] and can be seen in Figure 4.12. The y axis decides the layer for the row, with input at y=0 and output at y=4. It has 16 hidden neurons divided up into three layers. The first hidden layer have eight neurons, while layer 2 and 3 have only four neurons each. The x axis uses the middle as 0, so that the left and right side of the substrate are easier to separate for the CPPN. This substrate is designed to be similar to the HXOR problem, and the designer have the goals that it will from into multiple modules that each solve an XOR function, similar to what seen in the blue boxes in 4.13



(c) H-XOR

Figure 4.12: Handcrafted substrate used for the HXOR problem

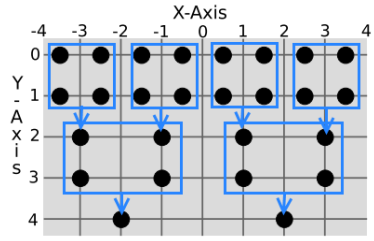


Figure 4.13: Handcrafted HXOR substrate are designed to be similar to the problem

RQ3	How does the evolved substrates perform on modular problems compared to manually designed substrates?
Test plan overveiw	Test if the evolved substrates is on par or better than hand-crafted substrates for achieving modularity. • **T4** Run without allowing substrate to evolve and with an initialized substrate for the tasks which are chosen from well performing design in the literature. Then compare performance earlier tasks with results gained with evolved substrates from the previous experiments. • **Measure:** Average and maximum performance of the best individuals, and the standard deviation of these. • **T4.2** Test performance of the algorithm on HXOR • **Focus of analysis:** Compare results from T4 with earlier results in **T1**
Hypothesis	The Hypothesis is that the algorithm without substrate evolution will produce less modular ANN, and because HyperNEAT is sensitive to changes in the substrate it would be difficult/lucky to expect to hit upon a good design on the first try. For this reason, the hand-crafted substrates are expected to have a more difficulties solving the modular HXOR problem, and thus will show worse results than what was seen in T1.

4.6.2 T4: Results and Analysis

	Mean Acc	std.	Max Acc
Handcrafted substrate	0.94609375	0.01921	0.98046
T1 best(Classic)	1.0	0.0	1.0
T1 worst(MSS)	0.97890	0.03045	1.0

Table 4.7: T4: Achieved accuracy of handcrafted substrate and evolved substrates on the HXOR problem

The results shows that the algorithm running without being allowed to alter the substrate structure performed worse on the modular problem compared to results from T1, which can be seen in Table 4.7. On average standard HyperNEAT only achieved individuals that got around 94.6% performance on average, and it never managed to achieve a 100% correct solution. In addition to this the standard HyperNEAT never achieved a perfect solution to the HXOR problem. The CPPN adaption that was used for this experiment was also the one that consistently achieved 100% performance in T1 where the substrates were evolved. Even the worst CPPN alternative from T1 were able to achieve perfect solutions, and had an average accuracy of 97.9%, higher than that archived in this experiment by standard HyperNEAT using a handcrafted substrate. Since the handcrafted substrate were not able to achieve a perfect solution to this task, it means that it could not achieve F-modularity to the same extend as when evolution were responsible for the design of the substrate structure. It is possible that standard HyperNEAT could have performed better if evolved for more generations, but the low standard deviation indicate that all runs had flattened out on about the same fitness and probably were close to convergence. This confirms the hypothesis that evolving substrates grants a benefit when solving modular problems.

The standard HyperNEAT did however seem to be a bit more stable than the CPPN adaptions that could not consistently produce perfect solutions in T1. Table 4.7 shows that standard HyperNEAT had a standard deviation of 0.019, while the sub optimal CPPN adaptions from T1 had 0.03 when training was used. Another source of the stability compared to T1, could be that the substrate is locked in place and never mutates, and only the CPPN were evolving. This means that evolution only had one possible substrate to combine with each CPPN, while the EMSS-HyperNEAT could combine multiple substrates structures to the same CPPN. The low standard deviation means that it often ended up with very similar solutions, supporting the hypothesis that it is very dependent on the substrate geometry. If the substrate is fixed in place at the beginning

and can't evolve then the user has to be "lucky" with the chosen design of the substrate. Consequently lower variation are exhibited in standard HyperNEAT compared to EMSS-HyperNEAT if the worse CPPN versions are used.

T4 also shows that the use of machine learning in combination with the evolutionary algorithm proved to be more successful then either of them used separately for solving the HXOR problem. The EMSS-HyperNEAT were shown to not solve the HXOR problem without training in table 4.3.2, and the results in Table 4.7 also shows that HyperNEAT with BP, but without substrate evolution also were not able to solve the HXOR problem.

Chapter 5

Conclusions and Future work

This chapter presents a discussion of the EMSS HyperNEAT based on the results from the previous chapter, in the context of the thesis' research goal and research questions. Following this are the contributions of the research, before finally the future work is presented.

5.1 Discussion and Goal Evaluation

The overall research goal of this work have been to investigate how HyperNEAT can be extended to include evolution of the substrate with the goal of achieving modular ANNs. Towards this goal this several research questions was created, and experiments have been conducted to attempt to answer them. The resulting answers that were found to each research question are discussed below.

Research question 1 *How should the CPPN be adapted to achieve modular neural networks?*

Research question 1.2 *How should the CPPN outputs be adapted to achieve the highest performance on image classification accuracy?*

These first two research questions both regards which of the CPPN variations from section 3.6 that should be used when the substrates evolve:. The classical CPPN adaption achieved the best result both for solving problems designed to require modularity and on image classification. The MSS-CPPN version which

are the most usual to use in the literature with MSS, were actually the one that had the most problems with variance, and achieved the worst results both for F-modularity and in image classification. The risk of unfavourable correlations of neurons that was a motivation for the design of MSS does not seem be a problem for the Classical version when the substrate structure can evolve. Future work that uses evolves MSS should therefore consider using the classical CPPN version, since the more complicated implementation of the MSS-CPPN version is not guarantied to perform better.

Research question 2 *How does ANNs produced perform on problems with geometrical relationships or clearly consists of subproblems?*

Results from the experiments on HXOR, 5XOR and the retina problem show that EMSS-HyperNEAT are able to consistently achieve ANNs that can solve these modular problems. All these problems are design so that they strongly encourage or require a F-modularity to be solved, and thus by solving them it is a strong indication that the ANNs are F-modular.

The geometrical relationships between the sub problems were not clearly apparent in the substrates being developed. The substrates were in general very different from human designed substrates, and it can be difficult for humans to achieve an intuitive understanding of the reason behind the specific geometries being evolved. This can both be that the geometries being evolved are just hard to understand, or that the geometries don't reflect the geometries of the problem as much as it simply have a favourable interaction with the CPPN being evolved together with it.

Research question 3 *How does the evolved substrates perform on modular problems compared to manually designed substrates?*

Where the evolved substrates could consistently achieve perfect solutions to modular problems, this was not the case for manually designed substrates. Not all substrate structures can achieve perfect solutions for the HXOR problem, and without the ability to alter the substrate it was not consistently ended up with solutions that achieved very similar results. When the substrates were allowed to evolve the achieved ANNs had a more varied and higher performance since evolution could move the population away from sub optimal substrates. This strongly suggest that being able to search for substrates in addition to only the evolve the CPPN offers a clear benefit, but at the cost of more variation in both shape and performance of ANNs produced.

Limitations

The certainty of the experiments is held back by there only being done five runs of each configuration. This was done because of time and resource limitations. It would have been preferable if more runs could be done to increase the certainty of the results from the experiments. Additionally using more generations in the evolutionary runs would increase certainty that evolution had convergence on the maximum possible performance before the different methods were compared. Because previous work with CCT have had strongest effect on the modularity late in evolution as discussed in 2.2.1, the limited number of generations used in this work could have made it difficult to see the full effect of the new method of calculating connection cost.

Goal *Investigating how HyperNeat can be extended to include evolving the substrate so as to achieve modular networks*

In regard to the research goal, the project has successfully created a an algorithm called EMSS-HyperNEAT that are able to evolve substrates and trough experiments it has been shown to solve modular problems, giving a strong indication F-modularity were achieved. The algorithm was not able to achieve the same F-modularity without evolution of the substrate structure, showing that substrate evolution was an integral part of the achieved F-modularity.

5.2 Contributions

This master thesis have presented the EMSS-HyperNEAT algorithm for evolving the substrate structures used in HyperNEAT and showed that it can solve modular problems. The algorithm combines several state of the art methods such as LEO, MSS, CCT and BP. Evolving the substrates are showed to perform better than hand-crafted substrate that was not allowed to change during evolution, showing that substrate design can be entrusted to evolution.

To give an evolutionary pressure towards modularity, a new way of calculating the connection cost was developed. This connection cost combines both a simulation of physical constraints and an abstract measurement of T-modularity. The physical constraint uses the spatial information of neurons in the substrate structure, while the other provides a more direct pressure for a division of the ANN into topological modules.

The effectiveness of a single set of CPPN output nodes were shown to perform best both on modular problem and a image classification problem, outperforming the more complicated CPPN adaption usually used with MSS. The evolution of

substrates seems to prevent the population getting stuck with a sub optimal assumption of neurons correlations, something that can be a problem with fixed substrates.

The work also shows that networks with full connectivity are able to solve many of the tasks used in the literature to asses if F-modularity are achieved.

5.3 Future Work

While doing this project there surfaced a lot of questions and alternative design decisions that that would be interesting to explore. Some of these were outside of the scope, but mostly it was just not enough time during the master thesis to explore everything. The following is a list of possible focus for future work that could not be done within this project:

1. Comparison between Gausian seed and CCT

   Because few generations was shown to be needed, Gaussian seed discussed in 2.2.1 might be a good alternative when HyperNEAT is used with BP. Because fewer generations are needed, the risk of the Gaussian hidden nodes disappearing are not as great. It would therefore be interesting with a comparison between Gaussian seed and CCT.

2. Improve interference measurement

   The interference used for the connection cost from section 3.5.1 does not care about the number of neurons in the substrate or the strength of connections. Future work can attempt to improve this measurement by including these to give a more accurate view of how information moves in the network. EMSS-HyperNEAT uses a binary "affects or not affects" to see how much of the network influence a substrate, and this could probably be improved by looking at the amount of effect instead.

3. Allow multiple input substrates

   In section 4.3.2 it was discussed the possible advantages of allowing for mutations that can split input and output nodes into several substrates instead of only one. Adding this could allow modules in the ANN to only receive specific inputs relevant to that module, and thus improve modularity.

4. Investigate the effect of coordinates that don't impact topology

   The effectiveness of using a single set of CPPN outputs for many substrates could have been a result of having multiple neuron coordinates that did not affect the LEO values as discussed in section 4.4.3. Future work

could investigate the benefits of giving neurons coordinates that only affect weight and biases, and not the LEO output of the CPPN.

5. Explore the effect of reusing substrates

   Direct representation of the substrate means that it a substrate has to be reinvent ed each time it is needed. There might be a benefit of reusing substrates. This could be done either by allowing the same gene to be active in multiple different layers, have a special mutation that copies a substrate from one layer to another. The first would allow for reuse and the alteration of many layers with a single mutation, while the second solution would be easier to implement but would act more similar to jumping genes (transposons) than to any true indirect encoding.

6. Further investigate F-modularity in the evolved ANNs

   Another thing that could have further investigated if the ANNs had F-modularity is to try to find out what modules solve witch problem. If a module of a perfect solution is deactivated or removed, what functionality is lost? By looking at what functionality is lost one can know what functions were processed within that module. If only one functionality is lost, it would mean that the module only did calculations on that one sub problem.

   An interesting result is that there were found solutions to the modular problems that had fully connected substrate structure. A more thorough investigation of how these ANNs solved the problems might reveal something about how F-modularity can be encouraged within networks that don't show any strong T-modularity. It should also be investigated if these simply preform an input-output mapping. Possible methods for doing this would be to solve larger more complex modular problems and use a test set. Alternatively look at how an ANN networks process inputs for the smaller problems. This would probably be more difficult, but might reveal if the neurons inside substrates are separated into parts of different functional modules, as discussed in section 4.3.2.

7. Investigate the geometries that are evolved

   A thing that can be done is to investigate if the substrates that are discovered are an inherent aspect of the problem or simply happened to work well with the CPPN. This could be done by having two separate populations of CPPNs and substrate structures, and using co-evolution methods so that a substrate structure need to work well with multiple different CPPNs, instead of just one.

   Another alternative can be to lock the CPPN and only evolve the substrates. If the same shapes consistently were optimal for a given CPPN,

there would be grounds to say that the substrates look for very specific geometries in the substrate, or if many possible geometries can work for the same CPPN. An heatmap could be used to help see if the neurons most often would be placed in the same locations. If many very different substrates can work with a specific CPPN it could indicate that the geometries chosen by evolution are fairly random in nature. If on the other hand there only seem to be one substrate structure that are able to effectively use a given CPPN, it would suggest that there are some important interaction between that specific geometry and the CPPN.

8. Use EMSS-HyperNEAT to evolve CNNs

   In section 4.4.2 it was discussed how EMSS-HyperNEAT showed a large improvement over state of the art. In section 2.2.4 it was also discussed how earlier work have achieved significantly better performance when using HyperNEAT and BP to evolve CNNs, compared to when ANNs were used. It would be interesting to see if EMSS-HyperNEAT also would show a large improvement in image classification if it were used to evolve CNNs.

9. Estimate future learning gain to improve training time and performance evaluations

   Future work could use some of the alternatives for performance evaluation discussed in section 3.5. These would have less risk of accidentally selecting for premature convergence, and also might require less training per generation. If the dotted arrows depicted in Figure 3.4 were extrapolated, then evolution could always know if an individual had converged or would have improved with more training time. This could both be used to select against early convergence and the steepness can be used to estimate future performance, and thus less training can be used per individual. This could significantly speed up the algorithm as training is the most time intensive part.

At the end of the project there exist many interesting directions for future work. It is possible to investigate the interaction evolution have on other methods for encouraging modularity, interactions between evolution and BP, investigate the reason behind geometries being evolved, interactions between evolved coordinates and CPPNs, and a more detailed investigation of the functional modularity in the ANNs evolved by EMSS-HyperNEAT.

Bibliography

[1] Amer, M. and Maul, T. (2019). A review of modularization techniques in artificial neural networks. *Artificial Intelligence Review*, 52(1):527–561.

[2] Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2008). On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188.

[3] Buxhoeveden, D. P. and Casanova, M. F. (2002). The minicolumn hypothesis in neuroscience. *Brain*, 125(5):935–951.

[4] Carroll, S. B. (2001). Chance and necessity: the evolution of morphological complexity and diversity. *Nature*, 409(6823):1102–1109.

[5] Chklovskii, D. B. and Koulakov, A. A. (2004). MAPS IN THE BRAIN: What Can We Learn from Them? *Annual Review of Neuroscience*, 27(1):369–392.

[6] Clune, J., Beckmann, B. E., McKinley, P. K., and Ofria, C. (2010). Investigating whether hyperNEAT produces modular neural networks. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, page 635, New York, New York, USA. ACM Press.

[7] Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society B: Biological Sciences*, 280(1755):20122863.

[8] Dellaert, F. (1995). Toward a biologically defensible model of development.

[9] Freeman, M. and Gurdon, J. B. (2002). Regulatory Principles of Developmental Signaling. *Annual Review of Cell and Developmental Biology*, 18(1):515–539.

[10] Guimerà, R. and Nunes Amaral, L. A. (2005). Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900.

[11] Gurdon, J. B. and Bourillot, P. Y. (2001). Morphogen gradient interpretation.

[12] Høverstad, B. A. (2011). Noise and the Evolution of Neural Network Modularity. *Artificial Life*, 17(1):33–50.

[13] Huizinga, J., Clune, J., and Mouret, J.-B. (2014). Evolving neural networks that are both modular and regular. In *dl.acm.org*, pages 697–704.

[14] Huizinga, J., Mouret, J.-B., and Clune, J. (2016). Does Aligning Phenotypic and Genotypic Modularity Improve the Evolution of Neural Networks? In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference - GECCO '16*, pages 125–132, New York, New York, USA. ACM Press.

[15] Kashtan, N. and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773–13778.

[16] Kashtan, N., Mayo, A. E., Kalisky, T., and Alon, U. (2009). An Analytically Solvable Model for Rapid Evolution of Modular Structure. *PLoS Computational Biology*, 5(4):e1000355.

[17] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.

[18] Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization.

[McIntyre et al.] McIntyre, A., Kallada, M., Miguel, C. G., and da Silva, C. F. neat-python package for python. `https://github.com/CodeReclaimers/neat-python`.

[20] Mountcastle, V. (1997). The columnar organization of the neocortex. *Brain*, 120(4):701–722.

[21] Moussian, B. and Roth, S. (2005). Dorsoventral Axis Formation in the Drosophila Embryo—Shaping and Transducing a Morphogen Gradient. *Current Biology*, 15(21):R887–R899.

[22] Newman, M. E. J. (2004). Detecting community structure in networks. *The European Physical Journal B - Condensed Matter*, 38(2):321–330.

[23] Newman, M. E. J. (2016). Equivalence between modularity optimization and maximum likelihood methods for community detection. *Physical Review E*, 94(5):052315.

[24] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. PyTorch official homepage: `https://pytorch.org/`.

[25] Pugh, J. K. and Stanley, K. O. (2013). Evolving multimodal controllers with HyperNEAT. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*, page 735, New York, New York, USA. ACM Press.

[26] Risi, S., Lehman, J., and Stanley, K. O. (2010). Evolving the placement and density of neurons in the HyperNEAT substrate. In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10*, pages 563–570.

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

[28] Risi, S. and Stanley, K. O. (2012b). An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density, and Connectivity of Neurons. *Artificial Life*, 18(4):331–363.

[29] Schrum, J. (2018). Evolving indirectly encoded convolutional neural networks to play tetris with low-level features. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18*, pages 205–212, New York, New York, USA. ACM Press.

[30] Soldal, K. V. (2012). Modularity as a Solution to Spatial Interference in Neural Networks. *110*, (June).

[31] Sosa, F. A. and Stanley, K. O. (2018). Deep HyperNEAT: Evolving the Size and Depth of the Substrate Evolutionary Complexity Research Group Undergraduate Research Report. Technical report.

[32] Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162.

[33] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

[34] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

[35] Verbancsics, P. and Harguess, J. (2013). Generative NeuroEvolution for Deep Learning.

[36] Verbancsics, P. and Stanley, K. O. (2011). Constraining connectivity to encourage modularity in HyperNEAT. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*, page 1483, New York, New York, USA. ACM Press.

[37] Wagner, G. P., Pavlicev, M., and Cheverud, J. M. (2007). The road to modularity. *Nature Reviews Genetics*, 8(12):921–931.

[38] Watanabe, C., Hiramatsu, K., and Kashino, K. (2018). Modular representation of layered neural networks. *Neural Networks*, 97:62–73.