Master's thesis

NTNU
Norwegian University of Science and Technology

Kasper Maagerø Svendsen

Exploring DES-HyperNEAT for reinforcement learning

Master's thesis in Master of Science in Informatics
Supervisor: Pauline Haddow
June 2022

NTNU
Norwegian University of
Science and Technology

Kasper Maagerø Svendsen

Exploring DES-HyperNEAT for reinforcement learning

Master's thesis in Master of Science in Informatics
Supervisor: Pauline Haddow
June 2022

Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Kasper Maagerø Svendsen

Exploring DES-HyperNEAT for reinforcement learning

master project, spring 2022

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

Abstract

Brains in nature have been evolved over millions of years, the connectivity formed by neurons and synapses is complex and can exist of billions of neurons in a single brain. Neuroevolution is a field that creates and optimizes artificial neural networks by using evolutionary algorithms. The evolution of brains in nature heavily inspires the field. Layered deep evolvable substrate HyperNEAT (DES-HyperNEAT) is a neuroevolution method that evolves a layout of modules called substrates connected by paths and assembles them into an artificial neural network. The modularity of DES-HyperNEAT is an example of inspiration taken from the brains in nature. DES-HyperNEAT has previously outperformed relevant methods on classification problems, this thesis explores DES-HyperNEAT for reinforcement learning.

Brains in nature have complex connectivity and the location of the neurons matters. This concept of the locality is explored for DES-HyperNEAT to create a connection control for the artificial neurons in the assembled artificial neural networks. Another area of exploration is the addition of bias to the neurons in the assembled artificial neural network, a concept strong in the field of Deep Learning. How the concepts affect DES-HyperNEAT have been explored by comparison with related methods and a tailored reinforcement learning algorithm.

Sammendrag

Hjerner i naturen har utviklet seg gjennom millioner av år, koblingene av nevroner og synapser er komplekse og det kan være milliarder av nevroner i en hjerne. Neuroevolution er et fagfelt som lager og optimaliserer kunstige nevrale nettverk ved bruk av evolusjonære algoritmer. Evolusjonen av hjerner fra naturen er en stor inspirasjon til fagfeltet. Layered deep evolvable substrate HyperNEAT (DES-HyperNEAT) er en metode som bruker evolusjon til å utvikle et kunstig nevralsk nettverk ved bruk av moduler kalt substrates som er koblet sammen via paths. Mudulartiteten i DES-HyperNEAT er et eksempel på egenskaper som er hentet fra hjerner i naturen. DES-HyperNEAT har tidligere utkonkurrerte lignende metoder på klassifiseringsproblemer, denne avhandlingen vil utforske DES-HyperNEAT på et reinforcement learning problem.

Hjerner i naturen har komplekse koblinger og plasseringen av nevroner har betydning. Dette konseptet av plassering er blitt utforsket for DES-HyperNEAT for å lage kontrollerte koblinger for nevroner i det kunstige nevrale nettverket. Et annet utforsket området er muligheten for å legge til partiskhet i nevroner i det kunstige nevrale nettverket, et konsept som står sterkt i fagfeltet Deep learning. Hvordan disse konspetene påvirker DES-HyperNEAT er utforsket ved sammenlignet med relevante metoder og en spesifisert reinforcement learning algoritme.

Preface

This thesis is a result of a master's project in artificial intelligence at the Norwegian University of Science and Technology. The initial literature review was conducted during a Preparatory Project the fall of 2021, where most of the knowledge for chapter 2 and 3 was acquired. The implementation of the method, the design decisions and experiments was a part of the Master's project spring of 2022.

I would like to thank my supervisor Pauline Haddow for excellent guidance and detailed feedback thought the project.

Kasper Maagerø Svendsen
Trondheim, June 20, 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Artificial neural networks (ANNs) are inspired by the biological brain, connecting artificial neurons and synapses to create networks to solve complex problems. The field of neuroevolution takes inspiration from evolution in nature to create ANNs. By applying evolutionary algorithms to change the features of the artificial neural networks, the networks have been able to solve complex problems and even outperformed humansHausknecht et al. [2014]. NeuroEvolution of Augmenting topologies (NEAT) proposed in Stanley and Miikkulainen [2002] is a neuroevolution method. NEAT is evolving the structure of an artificial neural network and the weights, compared to traditional methods where the network structures are fixed and only the weights are trained. When NEAT are evolving networks the structure is initially small and usually only consists of input and output nodes, but by adding more nodes and connections between these the network grows bigger and more complex. By starting small the networks also remain compact and the solutions found will not be unnecessarily large.

Because NEAT adds nodes and connections gradually there is an upper bound where NEAT has problems finding feasible networks. This is because the changes made to the network will become small compared to the size and complexity of the network. The changes made will not be significant enough to make progress towards a better solution. The solution to this was the introduction of an indirect encoding of the network where the evolution did not change the nodes and the connection weighs directly. Hypercube-based NEAT (HyperNEAT) [Stanley et al., 2009] is an indirect method for creating ANNs. Compositional Pattern-Producing Networks (CPPNs) were introduced by Stanley [2007], and are networks that take advantage of different mathematical functions like Sine waves and the geometry in Gaussian functions to create an outcome with patterns. By having a fixed network structure within a two-dimensional space called a substrate, HyperNEAT uses a CPPN evolved by NEAT to assign the weights of the connections. The position of two nodes is sent into the CPPN and the output is the weight for the connection between the given nodes. The drawback of HyperNEAT is the need for setting a fixed network structure before running the evolution, which for more complex problems can be a challenging task.

Evolvable Substrate HyperNEAT (ES-HyperNEAT) Risi and Stanley [2012] is an extension of HyperNEAT removing the need to set a fixed network structure. The method only requires input and output nodes to be placed in a substrate. Then the method uses the output of the CPPN to determine the node position in the final ANN. ES-HyperNEAT starts from the input nodes and iteratively discovers more nodes by searching for places of high variance. Lastly, the output

nodes are connected with a final search developing the finished ANN. Because ES-HyperNEAT uses NEAT to evolve a single CPPN that might need to create more extensive networks, the single CPPN output needs to become complex to support the variation of weights needed for a complex ANN. This becomes a challenging task that evolution might not be able to find.

The introduction of multi-spatial substrate (MSS) [Pugh and Stanley, 2013] showed the benefit of having multiple substrates each with its own CPPN output. MSS had a fixed network structure like HyperNEAT and finding a structure that produces quality ANNs is challenging.

Deep ES-HyperNEAT (DES-HyperNEAT) was introduced by Tenstad and Haddow [2021], creating a framework that combines the node search from ES-HyperNEAT with multiple substrates. The framework showed promising results on Iris, Wine, and Retina [Tenstad and Haddow, 2021] datasets for classification. The framework has yet to be researched on a reinforcement learning problem where HyperNEAT and ES-HyperNEAT are commonly used.

1.2 Goals and Research Questions

The goal of the thesis is:

Goal *Investigates how DES-HyperNEAT can be extended to solve complex reinforcement learning problems.*

It is desired to test how DES-HyperNEAT performs in a complex reinforcement learning problem where HyperNEAT and ES-HyperNEAT are commonly applied. Two additional extensions will be added to DES-HyperNEAT.

The first extension is adding a bias to the developed ANN. By adding the bias to the developed ANN the network created could solve more complex problems by allowing each node to shift the activation function with a constant. The other extension will be adding Link expression output to the CPPN to try to exploit the geometric concepts of the substrates.

To accomplish the goal the following research questions are explored:

Research question 1 *How should bias be applied to DES-HyperNEAT to positively affect performance?*

Research question 2 *How do Link Expression Output's features benefit multiple substrates in DES-HyperNEAT compared to single substrates in ES-Hyper-NEAT?*

Research question 3 *How does DES-HyperNEAT perform compared to NEAT, HyperNEAT, and ES-HyperNEAT in a reinforcement learning environment?*

Research question 4 *How does the generic La-DES DES-HyperNEAT perform compared to a tailored state-of-the-art reinforcement learning algorithm?*

1.3 Structured Literature Review

This section will cover the structured literature review process, the search words, inclusion criteria, and quality criteria. The literature review phase was conducted initially in the fall of 2021 to gain the knowledge and understanding to be able to answer the research questions. The focus of the literature search was first broad to the neuroevolution field before the search was narrowed down to focus on articles relevant to the model. Tenstad [2020] has also been used as an inspiration for the state of the art structure. The outcome of the literature review is described in chapter 3.

The search term *hyperneat OR (neuroevolution AND deep) OR ("indirect encoding" AND network AND evolution)* has initially been used to find relevant literature. Additional literature was found both from cited and citing the initial literature. The publishing platforms and search engines used are *Google scholar*, *IEEE explorer*, *Research gate*, *ACM*, *SpringerLink* and *ScienceDirect*.

The inclusion and quality criteria have been used to narrow down the results from the initial search. The criteria used for this thesis:

IC1 *The study is focusing on evolving both the weights and the topology of an artificial neural network.*

IC2 *The method does not utilize gradient descent.*

IC3 *The method is evaluated in a reinforcement learning environment.*

IC4 *Empirical results are presented.*

QC1 *The aim of the research is precise.*

QC2 *The research reflects on the design choices and the consequences of those choices.*

QC3 *The research is compared to previous research.*

Chapter 2

Background Theory

This chapter contains explanations of concepts and methods needed to understand the rest of the thesis. The explanation of machine learning is covered to understand the problem domain, while the background leading to HyperNEAT is covered to understand fundamental parts of the model.

2.1 Machine Learning

Machine learning is a field in artificial intelligence with a focus on making machines "learn" instead of programming them for a specific task. Machine learning is usually split into three different categories: supervised learning, unsupervised learning, and reinforcement learning. The problem presented in this thesis is a reinforcement learning problem.

2.1.1 Reinforcement Learning

Reinforcement learning is training a model to take a sequence of actions. The model used in reinforcement learning is usually called an agent. There are no training data needed in reinforcement learning. Instead, the agent is rewarded for their action in an environment. The agent starts with no prior knowledge and takes random actions. The rewards can be both negative and positive based on the outcome of the action and the agent tries to maximize the total rewards. If the agent is playing a game the reward will be higher for a win than a tie, and higher for a tie than a loss. The agent uses this feedback to perform better in the future.

In reinforcement learning, the designer only sets the reward policy, based on the task the agent needs to solve. It is up to the agent to learn how to gather the most rewards by learning how to take action in the environment. For example, in a simple maze environment, the agent will have to take steps to reach a goal. To reach the goal the agent needs to cross multiple tiles, the tiles are assigned a predefined reward value, where the goal typically has a higher positive value while normal tiles will get a lower negative or no reward value. The agent starts by taking random actions and will gradually learn the rules to get better rewards. This way as the agent optimizes its action policy.

2.2 Artificial neural networks

Artificial neural networks (ANNs) are computer systems inspired by the structure of animal brains. The ANNs are a collection of connected artificial neurons, also called nodes. These connected nodes form a graph, from the input nodes to the output nodes. The nodes between input and output nodes are usually called hidden nodes. In the network, each connected neuron pair has an associated value, called weight. The output of a single neuron is shown in Equation 2.1 and is calculated as the sum of each of the incoming connection weights, w_i, multiplied by the input value for that connection, x_i, plus a bias related to the node in question, b. Then this goes through an activation function, ϕ.

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b) \tag{2.1}$$

Because all single neurons in an ANN are a function the whole ANN forms a function. When the number of connected nodes increases the complexity of the function also increases. Different activation functions are used in ANNs for different purposes and it is common to have a different function at the output nodes than in the rest of the network.

Although the only requirement for an ANN architecture only is a directed graph with input and output, the most common way in modern machine learning is to use a layered feedforward network. A layered feedforward neural network is a directed acyclic graph (DAG), going in one direction from the input layer to the output layer, through an eventually hidden layer. The advantage of creating layered networks is that they easily can be calculated in parallel.

2.3 Evolutionary Algorithms

Evolutionary algorithms are heuristic search methods based on Darwinian evolution. They are used to find a solution to optimization problems and do not require gradient information to find near optimum solutions. The domain that needs to be optimized is called the search space and the search space for different problems can vary drastically in size. A good representation of a solution can help navigation in the search space. The algorithms usually take advantage of parallelism to evaluate the population of solutions.

2.3.1 Genetic Algorithm

Genetic algorithms (GA) are inspired by the evolution of species, where the fittest individuals in a population are more likely to reproduce offspring. Reproduction in genetic algorithms are called crossover and combines trait from two or more parents. Even though the offspring got the traits from their parents the different combinations of traits can be promising. In addition to crossover GAs usually have the ability to mutate traits in individuals.

The population of a GA is a set of solutions, each individual represents a solution. It is the designer of the program who chooses how to represent the individuals in the population. The individuals consist of a genotype and a phenotype. The genotype is the information about the traits of the individual, and it is what is passed on to offspring. The phenotype is what is observable and can be evaluated by a fitness function, a function used to determine the fitness of individuals in an environment.

The process of a genetic algorithm starts with the initialization of the population. The initialization is usually chosen at random but it could also be predetermined by the designer. Figure 2.1 shows all the evolutionary steps in a genetic algorithm. The next step after initialization is to

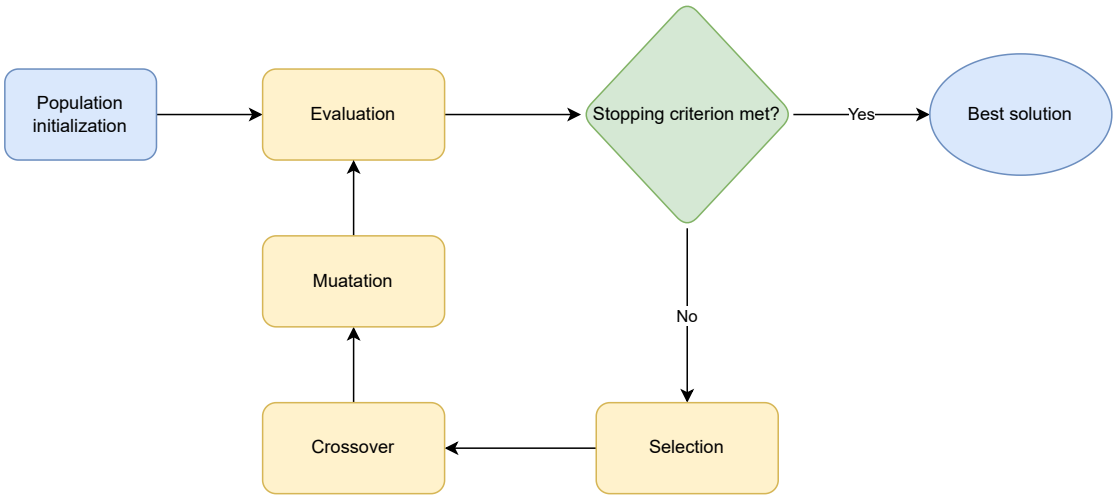


Figure 2.1: The process in genetic algorithms

evaluate the population using the fitness function. After the evaluation, the GA checks if a stopping criterion is reached. This can be a fitness threshold, a set number of generations, or a time limit. If the criterion is not reached the fitness from the evaluation is used to select the individuals for mating, this is usually simulating nature where the chance of being selected for mating increase with the fitness. Then crossover combines individuals from the population into offspring by taking traits from each parent. The process is usually random and there is no guarantee that the children will be more fit than the parents. The next step is mutation, a mutation is changing the traits of an individual to create variation in the population. A mutation is usually a small change to the genotype and the probability of how often it should happen depends on the choice of the designer. The goal of mutation is to search for different solutions close to what already exists. After mutation is finished the algorithm returns to evaluating and the cycle continues until the stopping criterion is met. When the algorithm terminates the algorithm returns the best solution.

2.3.2 Encoding

The population in an EA can either be direct or indirect encoded. In a direct encoding, the individuals directly represent the solution to a problem, there is no need to convert the genotype to a phenotype in these representations because the individual directly represents the solution.

In an indirect encoding, the genotype does not contain the solution by itself and therefore it needs to be developed into the phenotype. The development is a one-way mapping and each genotype can only represent a single phenotype. In this approach, the crossover and mutation are done on the genome and the conversion to phenotype is done at every generation of the evolution to calculate the fitness of the individual. An example of indirect encoding found in nature is DNA which is a genotype and biological development is the mapping to the phenotype.

2.3.3 Speciation

In nature, we divide creatures that share common traits and are able to reproduce with each other into species. These species share a finite amount of resources and the specie are therefore

limited in size. This concept can also be used within Evolutionary algorithms, where individuals with similar traits can compete with each other rather than the whole population.

To include speciation in evolutionary algorithms there is a need to compare how similar individuals are to each other, a distance measurement is usually used. The distance measurement can be between both genotype and phenotypes. To determine if two individuals are in the same species it is common to check if the distance metric is below a threshold.

2.4 Neuroevolution

Neuroevolution (NE) is inspired by the evolution of complex brains in nature, where evolution has produced brains with billions of neurons and trillions of connections. Instead of a physical brain neuroevolution evolves Artificial Neural Networks (ANNs) using evolutionary algorithms.

Because ANNs consist of both weights and a network structure NE methods are an umbrella term for all evolutionary methods that change some part of a neural network. Neuroevolution methods can have a fixed ANN and just evolve the weights or it can that evolve both weights and the topology of an ANN.

2.5 Neuroevolution of Augmenting Topologies

NeuroEvolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002] is a NeuroEvolution method that uses a GA to evolve the topology and weights of an artificial neural network. The method uses a direct encoding to represent the ANN. The network topology initially starts out minimally, but gradually evolves in complexity. Mutation in NEAT can change either the weights or the network topology or both.

The *Competing Conventions Problem* [Montana et al., 1989; Schaffer et al., 1992] is a problem that has more than one way of expressing a solution to an optimization problem with a neural network. When different encoded genomes represent the same solution, crossover is likely to produce poor offspring. NEAT solves this by using an innovation number for each connection gene.

The genomes (genotype) in NEAT are a list of connection genes and a list of node genes. Figure 2.2 show the genome and the ANN of an individual. A node gene includes a node key and type, the node can also include a bias. The connection gene consists of an in-node and an out-node, the weight, and an innovation number. The connection genes can be enabled or disabled by the evolutionary process, disabled connection genes will not show in the developed ANN. The innovation number is added when a connection is formed and is a global assignment. The number is used to match genes from different genomes during crossover and calculate the differences in individuals.

The development of the ANN is NEAT is fairly simple because of the direct encoding. The node genes are mapped directly to the nodes in the ANN and the connection genes with weights are included in the network if the gene is enabled.

Mutation in NEAT is done by changing weights in the connection genes, bias in the node genes, adding or removing a gene from one of the gene lists. When adding a new connection gene the two random nodes are chosen and the new connection gene is added to the list if the connection does not already exist. When adding a new node gene to the genome a random connection is chosen. The connection is disabled and two new connections form. The first connection is the connection between the old in-node and the new node, the weight for this gene is set to 1. The other connection formed is from the new node to the old out-node, the weight for this gene is the same as the weight of the old, now disabled connection gene.

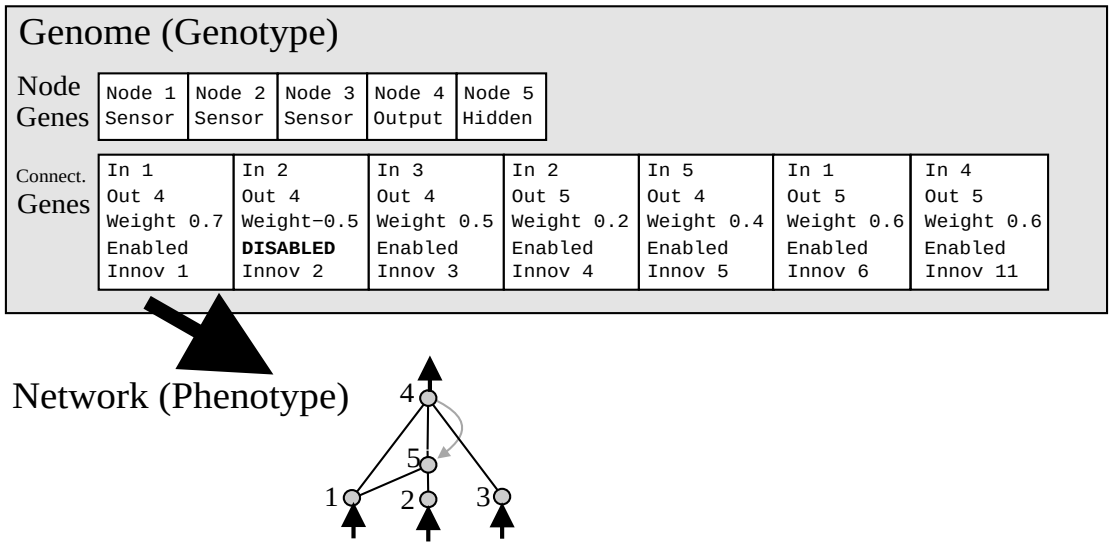


Figure 2.2: The mapping from genome to phenotype in NEAT [Stanley and Miikkulainen, 2002]

When using crossover on two genomes NEAT takes advantage of the innovation numbers previously described. When the innovation number for both parents matches the connection gene is randomly selected from one of the parents. When the genes do not match or are excess are chosen from the fittest parent.

The population in NEAT is divided into species based on similarities in the topology. Speciating the population in NEAT allows competition within species instead of across the whole population. Innovation numbers are used to divide the population, the more genes they share the more compatible the genes are. The compatibility distance, δ, shown in Equation 2.2, can be measured by disjoint, D, excess, E genes, and average weight difference of matching genes, $\overline{W}$.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \times \overline{W} \tag{2.2}$$

Where c_1, c_2, and c_3 are coefficients to adjust the importance of the factors and N are the number of genes in the larger genome, to normalize for the size.

The compatibility distance can then be used with a threshold δ_t to speciate the population. The spices are stored in an ordered list and that changes during evolution. For each generation, in the evolutionary process, the genomes are sequentially placed in a specie. A random genome from each of the species from the last generation is chosen to represent that species. For every genome g in the population, g is placed in the first species where the distance from g to the species representative is under the threshold, δ_t, if no such species exist a new species are created with this genome as a representative. The species uses explicit fitness sharing, where genomes of the same species share fitness within that species. This will discount the higher-performing species so other species have a chance to emerge.

NEAT initializes with just input and output nodes, instead of random typologies as some other neuroevolution methods uses. This way NEAT will have a small search space and incrementally increase the complexity by adding nodes and connections through mutation. This gives NEAT

the advantage in performance compared to other methods.

2.6 Compositional pattern producing networks

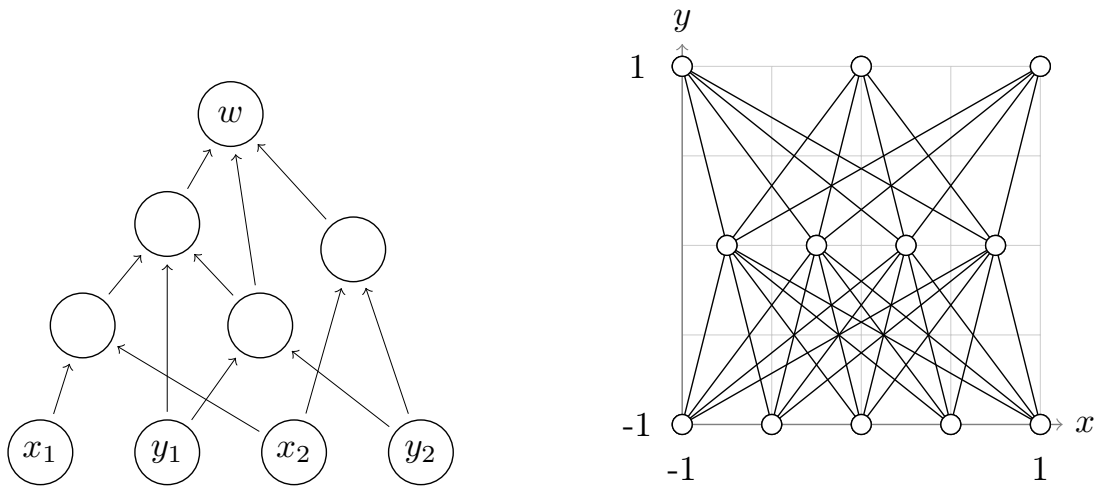
Compositional pattern-producing networks (CPPNs) [Stanley, 2007] are special types of neural networks designed to have an outcome that produces patterns. CPPNs differ from traditional ANNs because it uses a wide range of activation function in the same network. A CPPN consist of nodes and connection just like ordinary ANNs, but the activation functions used in the nodes can vary within the CPPN. The list of activation functions varies depending on the wanted features. Different activation functions produce different patterns, Gaussian functions produce symmetric patterns while a sine function produces periodic. The activation functions often used in CPPNs include, but are not limited to, sigmoid, relu, and tahn. The introduction of different activation functions allows the model to utilize properties of the different functions, and the output created tends to form more advanced patterns compared to a traditional ANN.

2.7 Hypercube-based NEAT

Hypercube-based NEAT (HyperNEAT)[Stanley et al., 2009] is a neuroevolution method that utilizes an indirect encoding for creating ANNs. HyperNEAT evolves CPPNs to assign weights to a fixed topology ANN. CPPNs are evolved using NEAT and in addition to the normal NEAT features, the activation functions in the node genes are evolved.

The topology of the ANN needs to be predefined in a two-dimensional space called a substrate. The substrate is a grid from -1 to 1 on both the x and y-axis. The ANN nodes are placed in the substrates before evolution. Figure 2.3b shows a substrate with five input nodes, four hidden and three output nodes. During the development of the ANN, the genomes are made into CPPNs with four input nodes and a single output. The input to the CPPN consist of x_1, y_1, x_2, and y_2 to represent the (x, y) position of nodes in the substrate while the output is the weight, w of the connection between the nodes. The CPPN, therefore, needs to be executed once for all connected nodes. If the magnitude of a weight w is under a set threshold a connection will not be formed. Figure 2.3a shows an example of an evolved CPPN. In addition to the four input nodes displayed in the figure, it is also possible to add the fifth node- This node is used as a bias node and will always input a set value to the CPPN.

As described in section 2.2 a bias in a the ANN is added to the sum of incoming connections. HyperNEAT has two ways of adding a bias to the ANN. One way is to add a bias node in the substrate and then use the CPPN output from this node to all other nodes as a bias. The CPPN then needs to set the weight for the connections and the bias for all nodes. The other way of adding bias to the ANN is to add a second output to the CPPN. The CPPN will then have two output nodes one for weight, w, and one for bias, b. The bias is then assigned to all nodes by executing the CPPN with the (x_1, y_1) position set to $(0, 0)$ and the $(x_2.y_2)$ set to the position of the (x, y) position of the nodes.



(a) CPPN **(b)** Substrate

Figure 2.3: HyperNEAT weight assignment. For each node pair (x_1, y_1), (x_2, y_2) in the Substrate (b) the CPPN (a) creates a weight, w. [Tenstad and Haddow, 2021; Pugh and Stanley, 2013]

Chapter 3

Related work

The following chapter is split into two sections. The first will cover a general state of the Art for neuroevolution with a focus on NEAT and HyperNEAT methods. The second section will cover a more detailed background of ES-HyperNEAT and DES-HyperNEAT for the purpose of extensions and experiments presented later.

3.1 State of the art

The following section covers the state of the art methods for Neuroevolution. subsection 3.1.1 covers Deep Learning, followed by network depth and complexity in subsection 3.1.2, and subsection 3.1.3 covers network encoding. subsection 3.1.4 and subsection 3.1.5 covers network connections and multiple substrates with a focus on HyperNEAT and its extensions.

3.1.1 Deep Learning

Deep Artificial Neural Networks (DNN) have in recent years had great success and even surpassed human performance in domains like strategic planning [Vinyals et al., 2019] and board games [Silver et al., 2016]. Proximal Policy Optimization (PPO) is a Deep Reinforcement learning method that uses an DNN to develop an action policy, the method utilizes gradient decent to optimize the neural network and has been able to solve complex RL problems [Vanvuchelen et al., 2020]. Gradient descent has been able to successfully optimize complex networks with millions of weights in the field of Deep Learning (DL) [He et al., 2016], but requires a differentiable error function. It is also normal to manually construct network topologies, requiring a manual trial and error search approach. Bias is also a great contribution to the correctness of predictions in networks Wang et al. [2019] that needs to be optimized. Even though networks from NeuroEvolution (NE) also have reached superhuman performance on specific tasks [Hausknecht et al., 2014], the networks are typically not as complex and deep as in DL. NE has, however, some desirable traits that are generally not found in DL. Many methods in NE do evolve the network topology during evolution, reducing the need for user design. NEAT and Evolutionary Acquisition of Neural Topologies (EANT) [Kassahun and Sommer, 2005] are NE methods evolving the topology and the weight of the network with no need for human experts. The search is broader than the single target search in gradient descent because the methods maintain a population of solutions.

Hybrid models have been proposed to utilize the benefits of DL and NE combined. Sun et al. [2020] used gradient descent to optimize a CNN that had a topology created by a Ge-

netic Algorithm (GA), commonly used in evolutionary methods. The networks generated by the method, called CNN-GA, performed comparably to the best human-generated network architectures. Without the need for human expertise, this highlights the power of having a population of solutions that can search broader than single gradient descent.

Without the need for human expertise in selecting a topology, no need for an error function and a broad search for solutions NE methods look superior to the alternative of gradient descent. However, the methods generally struggle with large search spaces and higher complexity [Such et al., 2017]. However, there are methods that are able to generate deep networks using evolutionary algorithms [Such et al., 2017].

3.1.2 Network depth and complexity

Deeper networks have the advantage of being able to represent more complex functions and therefore solve more complex problems. The performance of the 152-layered network produced by He et al. [2016] indicated that deeper networks can be more accurate than shallow networks. Neuroevolutionary methods like the NEAT algorithm struggles to create networks of this size as the search space becomes too large [Miikkulainen et al., 2019].

In addition to the 152-layered network, He et al. [2016] also produced an even deeper network with 1202-layers, this did not perform as well as the 152-layered network, implying that there is an upper bound where performance drops. He et al. [2016] argued that the deeper 1202-layered network performed worse because it causes overfitting because the problem did not require that much complexity. The 152-layered network performed better than both the shallower and deeper architectures it is likely closer to an optimal network depth. The challenge is therefore to find an optimal network complexity where the network understands the task but does not overfit. Miikkulainen et al. [2019] proposed CoDeepNEAT as a method that uses the gradual complexification from NEAT to find such a network by evolving topology, components, and hyperparameters and then training the network using backpropagation. The resulting network from their experiments is comparable to handcrafted methods, with the advantage of a more automated algorithm.

3.1.3 Network encoding

Because NEAT uses a direct encoding to represent the ANN, the isolated mutations make less impact the larger the ANN becomes [Gillespie et al., 2017]. NEAT, therefore, has problems with optimizing larger ANNs and are unlikely to be able to evolve deep and complex ANNs.

To create larger networks Stanley et al. [2009] introduced HyperNEAT, an indirect encoding for ANNs. section 2.7 describes the HyperNEAT method. Using indirect encoding it is possible to evolve more complex networks by decreasing the search space. Because of the compressed search space, the indirect encoding may be more suited for evolving deep and complex networks [Koutnik et al., 2010]. HyperNEAT uses a CPPN to assign weight to a static ANN topology. By allowing the evolving the CPPN instead of the ANN directly the mutations have a bigger impact.

3.1.4 Network connections

There are no limits on the amount connections in the static ANN topology created for HyperNEAT. Not all connections might be necessary and HyperNEAT, therefore, uses pruning to remove some of the connections. The connections are only included in the network if the absolute value of the weight, w from the CPPN is under a set threshold [Stanley et al., 2009]. By using pruning the resulting ANNs are more diverse by not always being fully connected.

As HyperNEAT uses the weight from the CPPN output, w to both assign weight value and prune the ANN the single output has two responsibilities. To separate these responsibilities Verbancsics and Stanley [2011] introduced Link Expression Output (LEO) to HyperNEAT (HyperNEAT-LEO). LEO is a separate output node added to the CPPN, and this node is used to prune the network. The networks include all connections with a LEO output over zero. This lets the weight freely change without the connections changing. With HyperNEAT-LEO it is possible to seed the CPPN with topographical principles by adding nodes connecting the input nodes to the LEO node with a Gaussian activation function. This increases locality and nodes will tend to form connections to nodes close in the substrate.

Another approach to limit network connectivity and increase locality is by adding a cost to each connection. Increasing the cost proportional to the square of the connection length nodes favors other nodes that are closer in the substrate. Huizinga et al. [2014] extended HyperNEAT with this technique making, HyperNEAT Connection cost Technique (HyperNEAT-CCT). Their experience showed that HyperNEAT-CCT was more modular, regular, and higher performing than HyperNEAT. The networks made were prioritizing connections of nodes closer to each other and made networks more similar to biological brains.

3.1.5 Adding more substrates

When drawing the weights to a crowded DNN using a single CPPN, the CPPN needs to be able to produce complex patterns. The connections are defined on an axis between -1 to 1 for each dimension based on the location of the connected nodes. When the number of connections in a substrate is in the thousands the distance between the nodes becomes minimal. The close nodes might need a big variance in weighting and the CPPN needs to produce sufficient weightings for all the nodes. The CPPN then might not be able to learn the complex patterns of the domain within the evolution.

Pugh and Stanley [2013] developed Multi-spatial Substrates (MSS) as an extension of HyperNEAT. By extending from a single substrate to multiple the method was able to outperform HyperNEAT in crowded networks. By putting input and output nodes that are unrelated in different substrates the network avoided the problem of close connections having similar weightings in crowded networks. In addition, the grouping of correlated nodes evolution finds good weights more easily. Nodes in MSS are still placed manually in different substrates and the connection are placed between the substrates. The weights are evolved in a single CPPN with multiple output nodes, one for each pair of connected substrates.

Pugh and Stanley [2013] used MSS to evolve a multimodal controller to an agent with different kinds of sensor inputs. By splitting the different sensor types into different substrates MSS removed the problem of manually placing unrelated input nodes in the same substrate. They suggested adding LEO or connection cost to complement the method by adding a more organic approach to modularity.

A similar extension of HyperNEAT is DeepHyperNEAT by Sosa and Stanley [2018]. Instead of a predefined network topology, DeepHyperNEAT separates itself from MMS by increasing the depth and breadth of the network by letting evolution add new layers to the substrate. The input and output nodes are placed in the substrate manually before the evolution starts with a connection in between. Each output node in the CPPN maps to the connection of two layers. During evolution mutation adds output nodes to the CPPN, adding more layers. The CPPN encodes an identity pattern between the newly added layer and the old connected layers to minimize the disruption in the output. In the experiments run by the researchers, the network was able to solve the XOR problem to validate the system's performance, and they suggest the method could give sufficient results on complex problems.

While the connection of the layers evolved during evolution the number of nodes within the added layers is predefined. Since the CPPN determines the weight of the connection based on the position of the nodes, the number of the nodes is essential for the algorithm to evolve good weights. Having the wrong number of nodes will poorly affect the performance and in the worst case, the network will not learn the necessary weights.

3.2 Relevant algorithms and frameworks

This section will cover a relevant algorithm and a framework for the purpose of extensions and experiments coming later. subsection 3.2.1 will cover the iteration search of Evolvable-Substrate HyperNEAT (ES-HyperNEAT) [Risi and Stanley, 2011] in detail. subsection 3.2.2 will cover the Deep ES-HyperNEAT framework [Tenstad and Haddow, 2021] with a focus on the Layered DES-HyperNEAT implementation.

3.2.1 Evolvable-Substrate HyperNEAT

The model Evolvable-Substrate HyperNEAT (ES-HyperNEAT) by Risi and Stanley [2012] is covered in this subsection. The model is an extension of the HyperNEAT method explained in section 2.7. In comparison to HyperNEAT, ES-HyperNEAT does not need a manually constructed network topology. Instead, ES-HyperNEAT discovers connections based on the pattern in the hypercube applied by the CPPN. The nodes used to form the connections are therefore also included.

Iterative Node Search

ES-HyperNEAT uses an iterative node search algorithm, called *Iterative network completion*. Each node search in the algorithm takes a single node position as an input and returns nodes that it should connect to as an output. The node search is two-phased. The first phase is called the *devision phase* and is shown in Figure 3.1. The division phase utilizes a quadtree structure to recursively split the substrate square, starting from the whole substrate into four new squares until a given resolution is reached (3.1b). Then the CPPN is queried for all the leaf nodes in the quadtree (3.1b) and the values are used to determine the variance for all higher nodes in the tree. The second phase in the iterative search is called *pruning phase* and uses a depth-first search to remove all tree nodes with a variance smaller than a given threshold (3.2a). In Figure 3.1 the grey node, 3, have a variance bigger than the threshold and the leaf nodes for node 3 are therefore kept after the pruning. The resulting leaf nodes are then checked if whether they are in a band. Nodes in a band are included, and the others are discarded. To determine if the node is in a band, the band value are calculated by the function, $\beta = max(min(d_{top}, d_{bottom}), min(d_{left}, d_{right}))$. If the band value, β is larger than a predefined threshold the points are included and a connection will be made from the node at the position where the search was performed from(3.2b).

Each iteration of the iterative node search performs a node search from all unexplored nodes, starting with the input nodes. The search continues for a predefined number of iterations discovering new hidden nodes for each iteration. An unexplored node will not connect to a node from a previous iteration to avoid cycles. After the iterative search, there is no guarantee that a path exists from the input nodes to the output nodes. Therefore one last search is done in reverse from the output nodes. This will not discover any new nodes. Then the network is cleaned by removing all nodes that are not connected to both the input and output nodes.

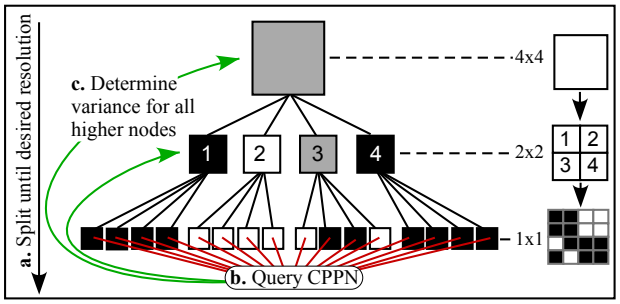


Figure 3.1: The division phase. [Risi et al., 2010]

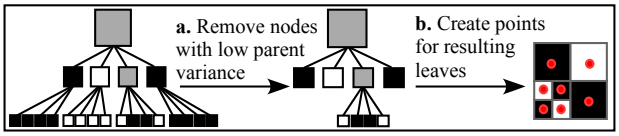


Figure 3.2: The pruning phase. [Risi et al., 2010]

ES-HyperNEAT LEO and Geometry seeding

As the CPPN used in ES-HyperNEAT does not differ from the traditional HyperNEAT, ES-HyperNEAT can be extended with LEO [Risi and Stanley, 2012]. The extension adds the advantage of determining the node density by implicit information in the CPPN. The extension does however only increase the performance when seeded with geometry and locality information [Risi and Stanley, 2012]. The geometry and locality seeding for ES-HyperNEAT is performed by adding two nodes with Gaussian activation functions between the input and the output nodes during initialization. Figure 3.3 shows a CPPN with such seeding. The G_1 takes $x_1 - x_2$ as an input and connects to the LEO node with a bias of -1, while G_2 takes $y_1 - y_2 + b$ as an input and connects to the weight output node. G_1 will therefore peak when the x_1 and x_2 have the same value and seeds the CPPN with locality along the x-axis. G_2 will create different weights based on the y position sin the substrate.

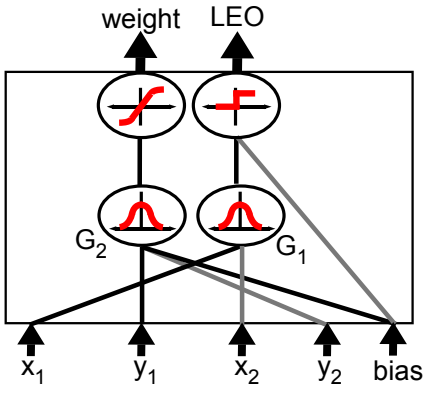
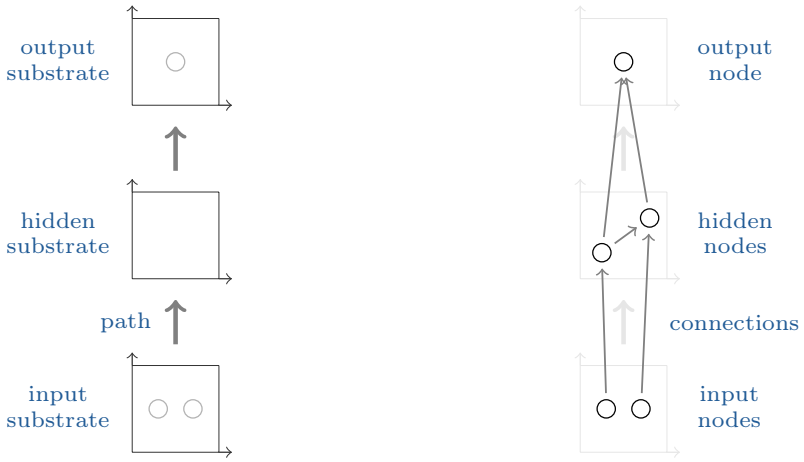


Figure 3.3: X-locality and Geometry seeding for a CPPN. [Risi and Stanley, 2012]



(a) Layout **(b)** Network

Figure 3.4: The difference between the Layout and the ANN network. [Tenstad, 2020]

3.2.2 Deep ES-HyperNEAT

This subsection describes the Deep ES-HyperNEAT (DES-HyperNEAT) framework presented by Tenstad and Haddow [2021]. DES-HyperNEAT is a framework rather than a method and this thesis will focus on the Layered DES-HyperNEAT implementation (LaDES). DES-HyperNEAT utilizes the iterative node search from ES-HyperNEAT to connect and explore multiple substrates.

Layouts

The substrates in DES-HyperNEAT are independent of each other and are connected to each other with *paths*. *Element* is a term used for both substrates and paths. A configuration of substrates and paths is called *layouts*. The layout evolved during evolution and is then used to *assemble* an ANN. The assembled ANN consists of *nodes* and *connections*, that are discovered during the assembly. It is important to distinguish between the two types because both will form a directed graph. Figure 3.4 shows the difference between a Layout and an ANN.

Overview

Before the evolution start, the layouts need to be I/O configured. Input and output nodes are placed in input and output substrates respectably during the configuration. Both input and output nodes can be grouped into a single substrate or divided into multiple. Once the I/O is configured the layout and CPPNs are initialized. Elements in DES-HyperNEAT each have a separate CPPN output in the layout. During assembly of the ANN, the CPPN outputs are used to perform the iterative node search inside the substrate and cross paths. The assembled ANN is then evaluated and if the stopping criteria are reached the best ANN is saved. If not the layouts and CPPN are evolved. Both the layout and CPPNs are evolved using the NEAT algorithm. An overview of DES-HyperNEAT is shown in Figure 3.5.

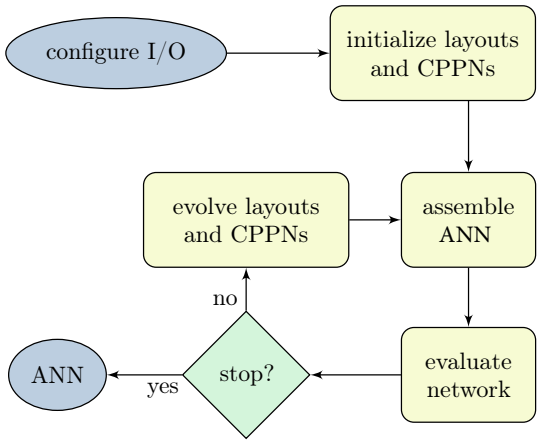


Figure 3.5: Overview of DES-HyperNEAT. Adapted from Tenstad [2020].

Network assemble

The network assembling in DES-HyperNEAT is done in a topological order where all elements are ordered. In Figure 3.6a the indicates the order of the element. First, an internal iterative node search is performed in the input substrate #1, but because the depth of the substrate is set to zero no nodes will be discovered. The substrate depth is evolved during evolution but the input and output substrates have a fixed zero depth. Input substrates are connected to substrate #6 by path #2 the node search is therefore performed from all nodes in substrate #1 using the CPPN output from path #2. When following a path only a single iteration of the node search is completed. The discovered nodes in substrate #6 are then created and a new node search is performed from all the nodes in substrate #1 to substrate #4 using the CPPN output from path #3. Then an internal search is performed in substrate #4 using the CPPN output for this substrate. The path #5 CPPN output is then used to connect to nodes in substrate #6 from all nodes in substrate #4, the node search can connect to previously created nodes or create new nodes. Another internal search is performed, this time in substrate #6 with three iterations. Then a node search from the nodes in #6 to the output substrate #9 with the CPPN output for path #7 and one from substrate #4 to the output substrate #9 with the CPPN output of path #8. Note that because the depth of the output substrate is set to zero no node searches will be performed internally in substrate #9 and only the connections to the existing node output will be present in the final network. At the end, the output substrate #9 will search in reverse to all the substrates with a path between them, and connect to existing nodes. Figure 3.6b shows the finished ANN. Because of the topological order of the search, no cycles will form.

Implementations

Tenstad and Haddow [2021] introduced three implementations of DES-HyperNEAT with the framework, Layered DES-HyperNEAT (LaDES), Single-CPPN DES-HyperNEAT (SiDES), and Coevolutional DES-HyperNEAT (CoDES). All implementation uses NEAT to evolve the layout but implements the CPPNs in different ways. LaDES uses multiple CPPNs, one for each substrate and path, the CPPNs are evolved alongside the layout for each genome. SiDES uses a single CPPN with one output node for each substrate and path, the CPPN are evolved together with the layout for each genome. CoDES uses one CPPN for each substrate and path, but the

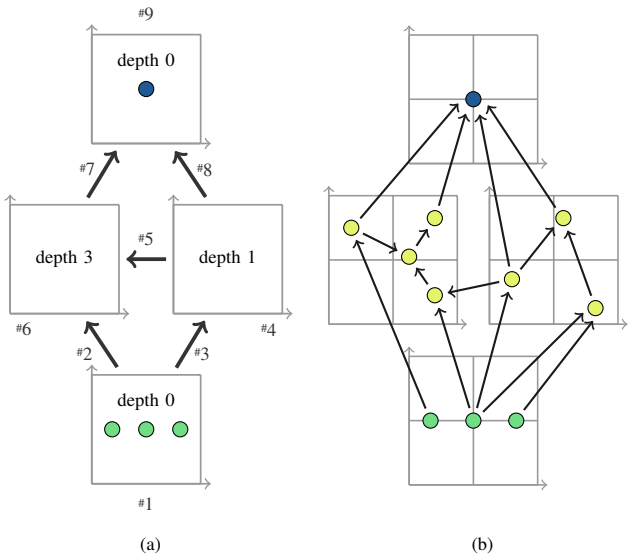


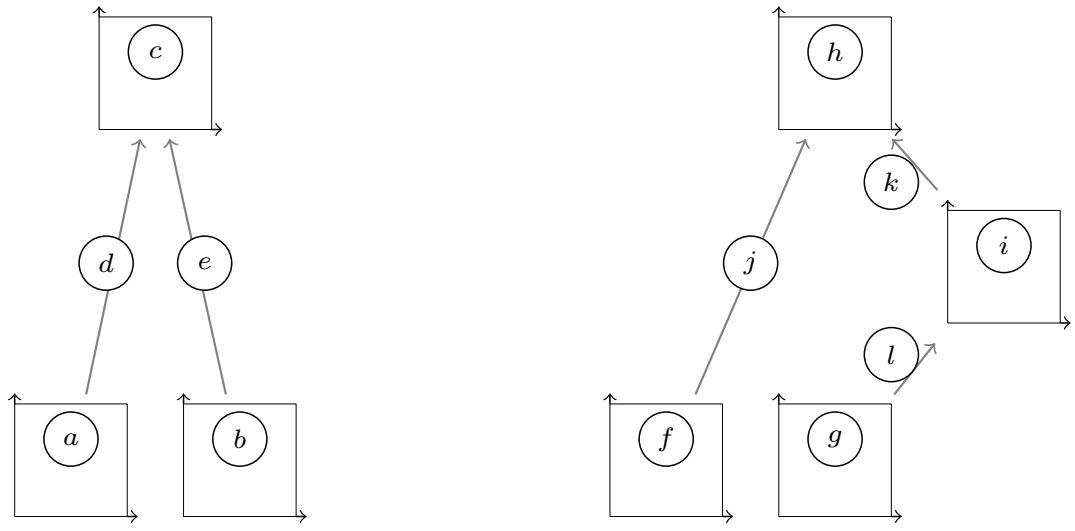
Figure 3.6: The assembly of an ANN. [Tenstad and Haddow, 2021]

layout and CPPN are split into two different populations. Using individual CPPNs for each element was shown to be highly advantageous and the LaDES implementation was performing better in all experiments performed. This thesis will therefore focus on expanding the LaDES implementation.

Layered DES-HyperNEAT uses individual CPPNs for each substrate and path in the layout. Figure 3.7 shows two layouts, the circles marked with *a-l* are unique CPPNs. The implementation is called Layered DES-HyperNEAT because it contains graphs in two layers, the layouts, and the CPPNs. The layout is evolved using NEAT and the genomes are layouts with a separate CPPN for each element. When new elements are added to a layout a new CPPN is initiated for the element. The elements also receive a unique id when created.

The elements are matched with id during crossover, this way crossover is only performed on CPPN within two matching path or substrates. In Figure 3.7 when crossover is performed on A and B, the CPPN c would crossover CPPN j as they are both in the substrate with the same id. The CPPN c would however never crossover with CPPN j or i as the element id would be different. Crossover is therefore combining both layout and CPPNs. Mutations will evolve both the layout structure and the CPPNs for the elements.

LaDES DES-HyperNEAT does, like NEAT, supports speciation to enable areas of the network to be optimized individually. The distance metrics used for speciation combine both the distance between the layout structure and the distance between their associated CPPNs. By mutating individuals new species will form.



(a) Layout A **(b)** Layout B

Figure 3.7: LaDES Layouts. All substrates and paths have an individual CPPN each marked a - l. [Tenstad, 2020]

Chapter 4

Model

The following chapter introduces the proposed extension of DES-HyperNEAT with a focus on reinforcement learning. The focus of the thesis will be on LaDES, as it previously was the best performing implementation of DES-HyperNEAT. Layered DES-HyperNEAT has previously only been applied to classification problems. The methods DES-HyperNEAT builds on, HyperNEAT and ES-HyperNEAT, previously have been successful at reinforcement learning (RL) problems. Therefore, this thesis will evaluate DES-HyperNEAT performance for an RL problem. The extensions added have a focus on increasing the performance for RL problems, but they are not exclusive to the domain. The extensions adds bias to the assembled ANN and the other adds an extra LEO node to each CPPN to let the CPPNs evolve more advanced connectivity.

The layout in DES-HyperNEAT is a graph that consists of substrates connected by paths. As the LaDES implementation is used, each element (substrate and path) have its own CPPN. Input and output substrates are manually created (I/O configuration), where input and output nodes are placed in the corresponding substrates. Multiple input and output substrates are allowed. The layout is the evolved using the NEAT algorithm to add substrate and paths and mutating the CPPNs. As the layouts are the basis for ANN assembly in DES-HyperNEAT, nothing is changed to the layout for this extension as RL problems also need assembled ANNs.

The evolved layouts are used to assemble the ANN during the network assemble phase. During the assembling of the ANNs, the layout elements are topologically sorted. The iterative node search algorithm, described in section 3.2.1, is used to discover node positions and connections both within (using substrate CPPNs) and across (using path CPPNs) substrates. Lastly, a reverse search is performed from the output nodes in the output substrate to all connected substrates. This extension of DES-HyperNEAT includes changes to the CPPNs, and therefore some changes are made during the iterative node search. The changes are described in section 4.1 and section 4.2. After the network assemble, the assembled ANNs can be evaluated, and if no stopping criterion is reached, the layouts can again be evolved to then assemble new ANNs. By splitting the evolution of the layouts and the assembly of the ANNs the mutations in layouts will have a bigger impact on the assembled ANN. This allows the ANNs to grow deeper than the directly mutating the ANN as in NEAT while still allowing a larger change in the weights than in dense single substrate alternative. Yet, the network will not grow deeper than needed as the layout will be evolved with NEAT and start out initially small.

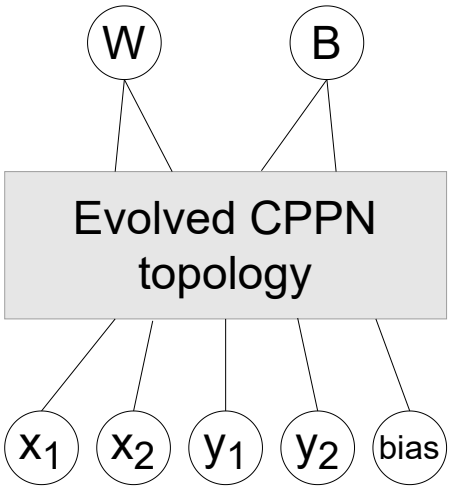


Figure 4.1: The CPPNs in the multiple CPPN bias variant of DES-HyperNEAT. The *bias* node is for bias in the CPPN, while the B node is to apply bias to nodes in the assembled ANN.

4.1 Introducing bias to the assembled ANN

Even though DES-HyperNEAT utilizes bias in the CPPNs (as shown in Figure 4.1), the assembled ANN does not contain any bias in the nodes. As bias has the ability to contribute to correct predictions in DNNs, as described in subsection 3.1.1, this would benefit DES-HyperNEAT. This extension adds two ways of adding bias to the assembled ANN in DES-HyperNEAT utilizing CPPNs.

4.1.1 Multiple CPPN bias

One way of adding bias to the assembled ANN is inspired by a method used by HyperNEAT, introduced in section 2.7. By introducing a separate output node in each individual CPPN, the node output can then be utilized to add bias during the discovery of new nodes. This variant will be referred to as the ***multiple CPPN bias*** variant. All CPPNs will then have two outputs, one for the weights and another for bias. An example CPPN is shown in Figure 4.1. When a new node is first discovered during the iterative node search, the output from the bias node, B, is used to assign the bias to the discovered node. If a node is discovered between substrates, the bias output from the path CPPN is used. The nodes will then receives bias based on when it is discovered. As there are multiple CPPNs to discover nodes, the biases may be evolved to suit a smaller group of nodes, but all CPPNs need to now be able to find good weights and biases. This will increase the search space for all the CPPNs.

4.1.2 Single CPPN bias

Another way of adding bias to the assembled ANN instead, instead of adding bias to all CPPNs in the layout, is to add a single bias CPPN to each individual in the population to evolve side by side with the layout. This variant of adding a bias CPPN will be called the ***Single CPPN bias***. The single bias CPPN will take the node position (x, y) as the input and output the bias, B. An example bias CPPN for single CPPN bias is shown in Figure 4.2. As the same bias CPPN is responsible for assigning bias to all nodes in every substrate, the nodes with the same

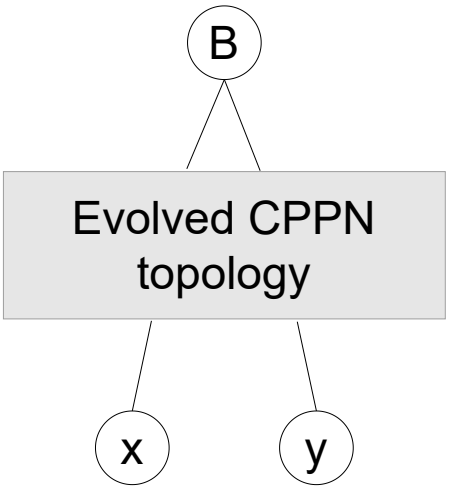


Figure 4.2: The CPPN that decides bias in the single CPPN bias variant of DES-HyperNEAT.

position in two different substrates will receive the same bias. The bias CPPN, therefore, needs to evolve to find biases that suit multiple nodes. As the CPPNs for all the substrates and paths are unchanged, no extra complexity is needed for the elements CPPNs to find weights and the only extra complexity added to the single CPPN bias variant is an extra CPPN.

4.2 Adding LEO and geometry seeding

Link expression output (LEO) has the advantage of separating the weight assignment and selection of which connections to include. As this has benefited both HyperNEAT and DES-HyperNEAT by selecting useful connections while also adding seeding to utilize geometric concept, as described in section 3.2.1. It is also thought to add benefits to DES-HyperNEAT by combining the advantage of both methods. By adding LEO nodes to all CPPNs in DES-HyperNEAT, the method should be able to evolve deeper modular ANNs (DES-HyperNEAT), while separating the chosen hidden nodes to a dedicated output node (LEO). The combined approach works as described in section 3.2.1. A dedicated LEO node is added to each CPPN in the layout. All connections discovered by the node search are kept if the output from the LEO node is greater than zero. As DES-HyperNEAT uses multiple substrates, will the LEO node be used to determine if connections should form both during node search internally in a substrate and when searching between different substrates. This will allow the LEO node to optimize to the specific elements, not just internally in a substrate.

Adding LEO nodes would also allow for seeding the CPPNs towards ANN structures. Therefore, in addition to the LEO node, this extension include the possibility, not a requirement, of adding seeding to each CPPN. The seeding is described in section 3.2.1. The addition of seeding will allow the geometric concepts to be introduced to the CPPNs.

4.3 Implementation details

The implementation was built using Python, even though DES-HyperNEAT previously have been implemented in Rust. This was chosen because of implementation of NEAT [McIntyre et al.,

Identity	x
Linear	$\begin{cases} 1.0, & \text{if } x \geq 1.0 \\ 1.0, & \text{if } x \leq -1.0 \\ x, & \text{otherwise} \end{cases}$
Step	$\begin{cases} 1.0, & \text{if } x > 0.0 \\ 0.0, & \text{otherwise} \end{cases}$
ReLU	$\begin{cases} x, & \text{if } x > 0.0 \\ 0.0, & \text{otherwise} \end{cases}$
Exp	e^x
Sigmoid	$\frac{1}{1+e^{-x}}$
Softplus	$0.2 * log(1+e^x)$
Tanh	$tanh(x)$
Gaussian	e^{-5*x^2}
OffsetGaussian	$2 * e^{-5*x^2} - 1$
Sine	$sin(x)$
Square	x^2
Absolute	$\|x\|$

Table 4.1: Activation functions implemented. Adapted from McIntyre et al. [2015]; Green [2003].

2015] and ES-HyperNEAT [Westh et al., 2017] previously have been successful, and Python are used for many reinforcement learning algorithms. The DES-HyperNEAT implementation has been built on top of python-neat library and the assembling of the network was built using some elements from the PUREPLES library. The simulator was adapted from Omelianenko [2019], and has been modified to match the domain described by Risi and Stanley [2012]. The activation functions used were adapted from earlier work [McIntyre et al., 2015; Green, 2003] and are presented in Table 4.1.

Chapter 5

Experiments and Results

In this chapter, the experiments to gain the knowledge to answer the research questions are conducted. In section 5.1 the introduction to parameters, hyperparameters, and results are given. Then the preliminary testing is presented in section 5.2 followed by the experiment plan in section 5.3. The experimental setup is given in section 5.4. Then section 5.5 - 5.8 are the experimental results and analysis.

5.1 Introduction

This section gives a brief introduction to the way parameters and hyperparameters are obtained and describes how the results will be presented.

In Table 5.1 there are given an example of an experimental setup. All rows are experimental parameters. The *methods* parameter is covering what methods will be evaluated in the environment set by the *domain* parameter. The *stop criterion* can be a set number of generations, evaluations, or a time limit. This will be repeated as many times as stated in the *repeats* parameter. The parameters will be presented in each experiment section.

To make a complete grid search of all parameter are infeasible for a method with the mount of hyperparameters that DES-HyperNEAT has. Therefore, it is chosen to adopt hyperparameters from earlier research to decide what hyperparameters are going to be in the hyperparameter search. Hyperparameters that are related are grouped together in the search. This is done to reduce the scope of the search further by only comparing relevant hyperparameters against each other. In the hyperparameter search, each combination is given a stopping criterion of 120 seconds to evaluate the combination. The hyperparameters with the highest impact are first found and are then fixed for the next search to reduce the amount of testing needed.

Results are presented with plots, tables and graphs. All experiments are performed multiple

Experiment X	
method	[NEAT, HyperNEAT, ES-HyperNEAT, DES-HyperNEAT]
domain	Hard maze
stop criterion	120 seconds
repeats	20

Table 5.1: Example of experimental parameters

	NEAT	HyperNEAT	ES-HyperNEAT	DES-HyperNEAT
population size	100	100	100	100
survivor ratio	0.2	0.2	0.2	0.2
add node prob	0.02	0.02	0.02	0.025
add connection prob	0.03	0.03	0.03	0.4
remove node prob	0.02	0.02	0.02	0.02
remove connection prob	0.01	0.01	0.01	0.1
weight mutation rate	0.94	0.94	0.94	0.94
mutate bias prob	0.7	0.7	0.7	0.7
mutate activation function		0.5	0.5	0.5
division threshold			0.3	0.05
variance threshold			0.03	0.5
band threshold			0.3	0.0
add substrate prob				0.05
remove substrate prob				0.002
add path prob				0.4
remove path prob				0.05

Table 5.2: Hyperparameters applied in the experiments.

times and graphs shown are an average over all runs. The t-test is used to determine the p-value where it is presented. When $p < 0.05$, the results are regarded as significant.

5.2 Preliminary testing

The work in Tenstad and Haddow [2021] identified an optimized set of hyperparameters to DES-HyperNEAT for classification. During preliminary testing, these parameters have been used as a foundation and compared with hyperparameters from previous NEAT, HyperNEAT and ES-HyperNEAT for reinforcement learning problems to find an optimal set of hyperparameters for DES-HyperNEAT for reinforcement learning. As the input and output nodes need to be placed inside substrates, different placements have been tested, both with single and multiple substrates for both input and output node placement.

For the other methods, ES-HyperNEAT, HyperNEAT, and NEAT parameters from previous work were used as a foundation. Because the population size was set higher in other experiments compared the ones performed here, a hyperparameter search was performed to explore adjusted mutation and crossover rates. The placements of input and output nodes inside the substrates have been taken from Risi and Stanley [2012].

The hyperparameters search performed resulted in the hyperparameters given in Table 5.2 and the activation functions are given in Table 5.3. These are the hyperparameters that will be applied in the experiments.

The resulting substrate configuration for each method can be seen in Figure 5.1. The green points are input nodes, the yellow points are hidden nodes, and the blue points are the output nodes. For HyperNEAT and ES-HyperNEAT, the substrate configuration used are from Risi and Stanley [2012].

Activation function	NEAT	HyperNEAT	ES-HyperNEAT	DES-HyperNEAT
Identity	No	Yes	Yes	Yes
Linear	No	Yes	Yes	Yes
Step	No	No	No	Yes
ReLU	No	No	No	Yes
Exp	No	No	No	Yes
Sigmoid	Yes	Yes	Yes	Yes
Softplus	No	No	No	Yes
Tanh	No	No	No	Yes
Gaussian	No	Yes	Yes	Yes
OffsetGaussian	No	No	No	Yes
Sine	No	Yes	Yes	Yes
Square	No	No	No	Yes
Absolute	No	Yes	Yes	Yes

Table 5.3: Activation functions.

(a) HyperNEAT (b) ES-HyperNEAT (c) DES-HyperNEAT

Figure 5.1: The substrates configurations used in the experiments.

5.3 Experimental Plan

To answer the research questions, four experiments have been conducted. Each experiment builds on the previous experiment. A description of each experiment is given below.

Experiment 1: Evaluate the two ways of adding bias to the assembled ANN in DES-HyperNEAT

Determine which, if any, of the bias variants are beneficial for DES-HyperNEAT.

Experiment 2: Compare the benefits of LEO, with and without seeding, for DES-HyperNEAT and ES-HyperNEAT

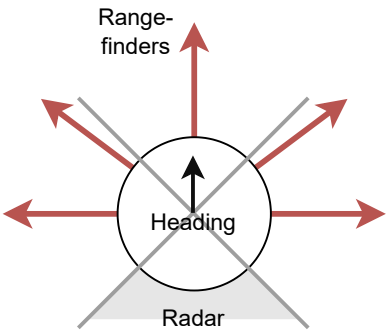
Investigate how LEO impacts the performance of the multi substrate DES-HyperNEAT compared to the single substrate ES-HyperNEAT. Both LEO with and without seeding will be compared for both methods.

Experiment 3: Compare the performance of the refined DES-HyperNEAT to NEAT, HyperNEAT, and ES-HyperNEAT on a reinforcement learning problem

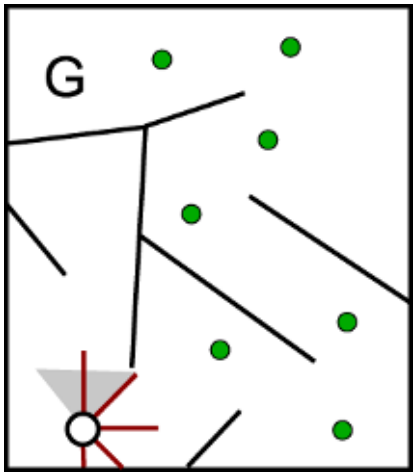
Gain the knowledge of how the refined DES-HyperNEAT performs compared to earlier hyperNEAT and NEAT methods.

Experiment 4: Compare the refined DES-HyperNEAT to Proximal Policy Optimization

Investigate the performance of the refined DES-HyperNEAT with the state of the art reinforcement learning algorithm Proximal Policy Optimization.



(a) The agent. Adapted from Risi and Stanley [2012].



(b) The maze.[Lehman et al., 2008]

Figure 5.2: The hard maze and the agent used in the experiments.

5.4 Experimental Setup

All experiments are performed on an Intel Core i7-7700. All evaluations of networks using NEAT and HyperNEAT-based methods have been run in parallel using 4 cores. The default hyperparameters used in all experiments can be found in Table 5.2. Any changes to the default parameters are highlighted in the individual experiments.

The experiment goal is to evolve a controller for an agent in a maze navigation domain, the maze experiment was first introduced by Lehman et al. [2008]. The agent is trying to reach a goal, G, by navigating a maze of walls. The agent has five rangefinder sensors to indicate the distance to the nearest wall. The rangefinders can be seen as red arrows from the agent in Figure 5.2a. The agent also has four pie-slice radar sensors that detect the general direction of the goal. There are three inputs to control the agent, left, L, forward, f, and right, R. The agent turns $(L - R) * 18°$ each action and moves forward $f * 20$ for each action of the simulation. The agent setup with sensors is adapted from Risi and Stanley [2012].

The environment chosen is the deceptive hard maze first presented by Lehman et al. [2008] and is shown in Figure 5.2b. The fitness of the agent is given based on how close the agent is to the goal at the end of the run. As the maze is deceptive, the agent is rewarded for discovering stepping stones, called *waypoints* shown in green points in Figure 5.2b, towards the goal. If the agent manages to get to the goal, extra fitness is received. The fitness function is given in Equation 5.1 as in Risi and Stanley [2012], where n is the number of waypoints passed and d is the distance to the next waypoint scaled to the range $[0, 1]$.

$$f = \begin{cases} 10, & \text{if agent reach goal,} \\ \text{n + (1 -d),} & \text{otherwise.} \end{cases} \tag{5.1}$$

The substrate configurations are shown in section 5.2. For HyperNEAT and ES-HyperNEAT the rangefinder sensor inputs are put at y coordinate -1.0 and the radar input is at y coordinate -1.2. For DES-HyperNEAT are the sensor types split into different input substrates.

	Experiment 1
DES-HyperNEAT variant	[Normal, Multiple CPPN bias, Single CPPN bias]
domain	Hard maze
stop criterion	350 generation or solved maze
repeats	20

Table 5.4: Experimental parameters for experiment 1.

5.5 Experiment 1: Adding bias to the assembled ANN

Experiment 1 evaluates the two different variants of adding bias to the assembled ANN, multiple CPPN bias and single CPPN bias, described in section 4.1. The parameters for the experiment are given in Table 5.4.

Hypothesis 1: The multiple CPPN bias variant will produce more complex individual CPPNs than in the normal and single CPPN bias variant.

Hypothesis 2: The multiple CPPN bias variant will produce more accurate biases than the single CPPN bias as the bias is more specific for each substrate or path.

Hypothesis 3: Both bias variants will use more time each generation compared to the normal variant due to the need to assign bias to all nodes in the assembled ANN.

Hypothesis 4: The variants utilizing biases will form less complex ANN networks as the bias will decrease the need for as many weights in the ANN.

The results of the experiments are presented in Figure 5.3. The fitness is used for comparison as it is an indicator of how far an agent traverses in the maze. The final fitness of the champion is used in the boxplot. The champion is the fittest individual in a population.

DES-HyperNEAT with multiple CPPN bias performs significantly ($p < 0.05$) worse than normal DES-HyperNEAT while there are no significant differences between Normal DES-HyperNEAT and DES-HyperNEAT with single CPPN bias. This indicates that hypothesis 2 is incorrect as the decrease in performance compared to normal DES-HyperNEAT is significant for the multiple CPPN bias variant. This might be because the additional complexity needed from each CPPN generally makes it harder to optimize.

The total number of connections in the CPPNs across the layout is significantly higher ($p < 0.05$) for DES-HyperNEAT with multiple CPPN bias than the other two variants. However, there are no significant differences between the number of hidden CPPN nodes in any of the variants. The mean number of connections and nodes in the CPPNs are shown in Table 5.5. This show an increase in complexity in the form of the number of connections when an additional CPPN output node is present, proving Hypothesis 1 correct. The total number of ANN nodes and connections are quite stable across the variants, with no significant difference between the normal DES-HyperNEAT and the versions with bias proving Hypothesis 4 incorrect. As all variants are not making ANNs that are not fully functioning controllers, the benefit of adding bias to reduce the complication of the weights might not be present.

Figure 5.4 show boxplots of the time used by each variant of DES-HyperNEAT. The normal variant of DES-HyperNEAT used slightly less time than the layout variant, but not significant. The variant with multiple CPPN bias did, however, use significantly ($p < 0.05$) more time than both the single CPPN bias and the normal version. Proving that Hypothesis 3 was incorrect because only the multiple CPPN bias variant was significanty slower.

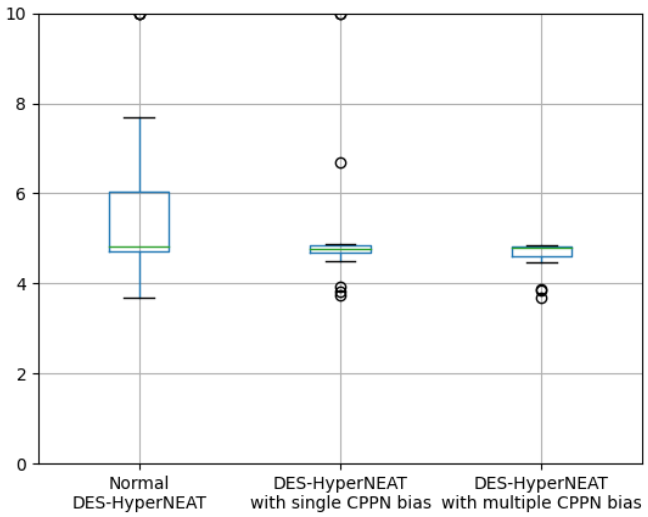


Figure 5.3: The champion fitness for the different variants of DES-HyperNEAT over 20 runs.

	DES-HyperNEAT	DES-HyperNEAT with single CPPN bias	DES-HyperNEAT with multiple CPPN bias
CPPN connections	28.80	30.25	58.05
CPPN nodes	1.25	1.40	1.45
ANN connections	34.45	33.45	43.75
ANN nodes	5.55	5.75	7.35

Table 5.5: The mean number of connections and nodes in the CPPNs and assembled ANNs of DES-HyperNEAT.

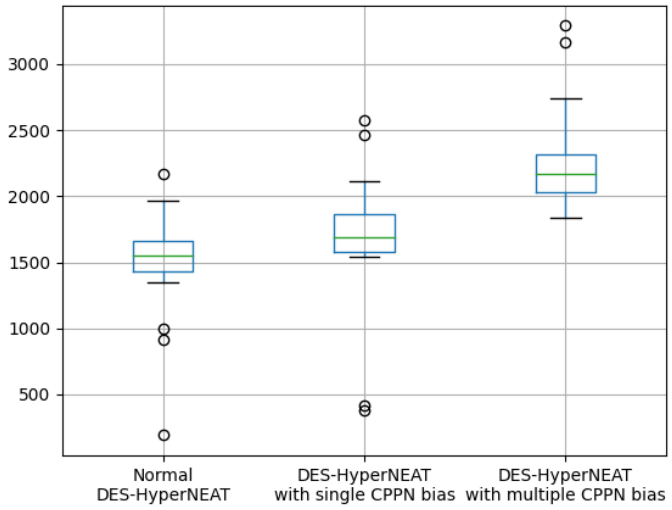


Figure 5.4: The time used for the different variants of DES-HyperNEAT over 20 runs.

It is concluded that adding a node to all CPPNs in DES-HyperNEAT with multiple CPPN bias is not beneficial. As there was no significant difference in the normal variant of DES-HyperNEAT and the single CPPN bias variant the seems to be no advantage of adding bias by a separate CPPN either. Further extensions are needed to see if any way of adding bias to the assembled ANN could increase the performance.

5.6 Experiment 2: LEO

Experiment 2 compares the benefit of LEO in single substrate ES-HyperNEAT and the multi-substrate DES-HyperNEAT as LEO has shown advantageous to ES-HyperNEAT in the past. This experiment has been split into two parts. First LEO will be added to ES-HyperNEAT, both with and without seeding, then LEO will be added to DES-HyperNEAT, also with and without seeding. The experimental parameters can be found in Table 5.6 for the first part of the experiment and Table 5.7 for the second part. The hard maze is used to compare the methods and variants, and all variants have been run a total of 20 times for a sample size. The normal DES-HyperNEAT has been used as none of the bias variants was significantly better and the goal of this experiment is to test the benefit of LEO extension to DES-HyperNEAT. The hypothesis are:

Hypothesis 1: LEO will decrease the number of connections in the best ANNs in both ES-HyperNEAT and DES-HyperNEAT because the CPPN(s) will include less connections during the node search.

Hypothesis 2: The fitness will increase with the addition of LEO with geometric seeding in ES-HyperNEAT and DES-HyperNEAT as the CPPN(s) have more control over the ANN connections.

	Experiment 2.1
ES-HyperNEAT variant	[Normal, with LEO, with LEO and seeding]
domain	Hard maze
stop criterion	350 generation or solved maze
repeats	20

Table 5.6: Experimental parameters for experiment 2.1.

	Experiment 2.2
DES-HyperNEAT variant	[Normal, with LEO, with LEO and seeding]
domain	Hard maze
stop criterion	350 generation or solved maze
repeats	20

Table 5.7: Experimental parameters for experiment 2.2.

Hypothesis 3: Both LEO variants will increase the running time of both algorithms as the CPPNs get more complex.

The results for the ES-HyperNEAT variants are shown in Figure 5.5. Both LEO variants of ES-HyperNEAT performed slightly but not significantly better than the normal variant. The boxplot shows the spread is wider for the LEO variants because some runs achieved a higher fitness, and both variants have an outlier that got a fitness score over 6, showing the potential that the LEO variants might climb better in the fitness landscape. The performance of ES-HyperNEAT variants with LEO, both with and without seeding, was very similar as seen in the Figure 5.5. The results of the DES-HyperNEAT variants are shown in Figure 5.6. As there was no significant increase in fitness Hypothesis 2 can not be answered without further testing. The number of ANN connections decreased significantly ($p < 0.05$) for both variants with LEO compared to Normal ES-HyperNEAT, and the mean value of nodes went down from 128 to 66 and 78, as shown in Table 5.8, with the LEO and LEO with seeding versions respectively. This shows that Hypothesis 1 was correct for ES-HyperNEAT.

The Normal version of DES-HyperNEAT performed significantly ($p < 0.05$) better than both the LEO variants. None of the LEO variants was able to get fitness as good as the 25% best runs of Normal DES-HyperNEAT. Hypothesis 2 is therefore incorrect as the LEO output nodes were shown to decrease the performance of DES-HyperNEAT. The number of ANN connections did not change significantly, and the mean number of ANN connections increased slightly without seeding and increased slightly with seeding. Hypothesis 1 is shown to be incorrect for DES-HyperNEAT.

The mean execution time of 350 generations of each ES-HyperNEAT and DES-HyperNEAT variant is given in Table 5.8. The difference in the efficiency of adding LEO to ES-HyperNEAT was insignificant and the mean execution time of the LEO variant was slightly lower than the normal ES-HyperNEAT. The variant with seeding, however was significantly ($p < 0.05$) slower than the other two variants. DES-HyperNEAT was significantly ($p < 0.05$) slower with the extension of LEO. DES-HyperNEAT with LEO used was over 60% more time on average and with LEO and seeding used twice as much time. Hypothesis 3 was, therefore, partly correct as there was no significant additional running time in ES-HyperNEAT, but DES-HyperNEAT ran significantly slower with the LEO extension.

Results show that DES-HyperNEAT does not benefit from the addition of LEO even though ES-HyperNEAT does. DES-HyperNEAT does not need LEO to choose more accurate connec-

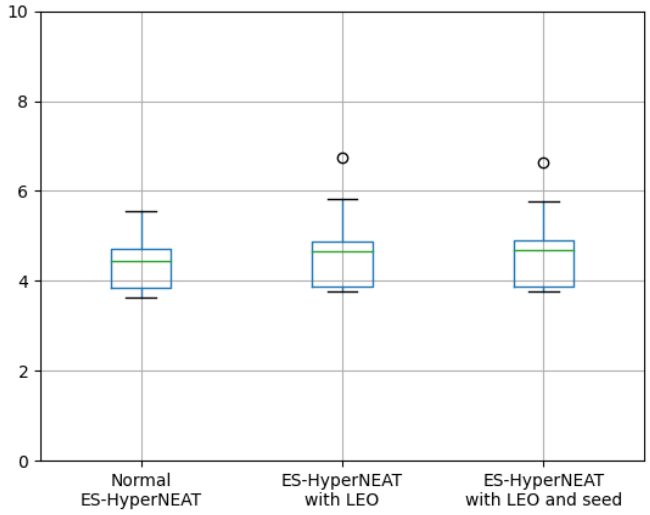


Figure 5.5: The champion fitness for 20 runs with the tree ES-HyperNEAT variants.

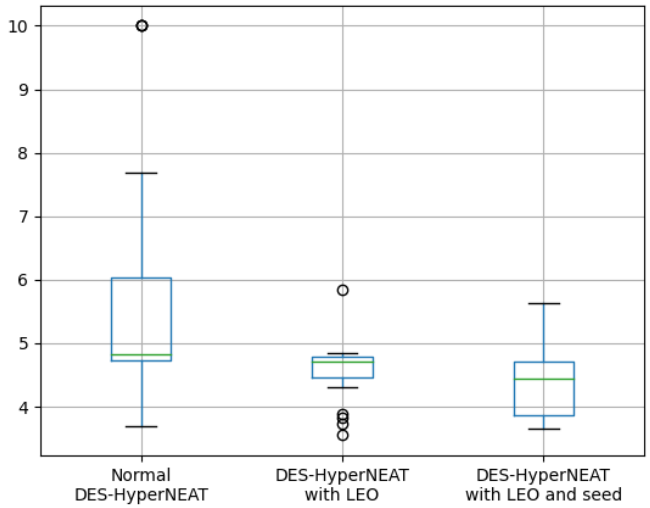


Figure 5.6: The champion fitness for 20 runs with the tree DES-HyperNEAT variants.

Variant	Mean time	ANN connections
ES-HyperNEAT	1199	128.25
ES-HyperNEAT with LEO	1184	65.85
ES-HyperNEAT with LEO and seeding	1288	72.65
DES-HypperNEAT	1483	34.45
DES-HyperNEAT with LEO	2465	38.40
DES-HyperNEAT with LEO and seeding	3474	27.80

Table 5.8: The mean execution time and number of ANN connections in seconds of 350 generations of the ES-HyperNEAT and DES-HyperNEAT variants.

	Experiment 3
methods	[NEAT, HyperNEAT, ES-HyperNEAT-LEO, DES-HyperNEAT]
domain	Hard maze
stop criterion	350 generation or solved maze
repeats	20

Table 5.9: Experimental parameters for experiment 3.

tions, and the extension makes optimizing the CPPNs overly complicated and adds significantly ($p < 0.05$) more running time to the algorithm. This is likely because of the already existing modularity in DES-HyperNEAT. When ES-HyperNEAT needs a dense network inside a single substrate, may DES-HyperNEAT instead create new substrates. This way all connections can be kept as the density of the nodes does not need to be high.

5.7 Experiment 3: Comparing related methods

Experiment 3 compares DES-HyperNEAT to other NEAT and HyperNEAT methods in the reinforcement learning problem presented. The methods compared are NEAT, HyperNEAT, ES-HyperNEAT with LEO, and DES-HyperNEAT. ES-HyperNEAT with LEO was chosen because of the slight increase in performance in experiment 2 without any significant loss in efficiency. Table 5.9 show the experimental parameters for experiment 3. Hypothesis 1 is formed based on earlier work with the maze navigation domain.

Hypothesis 1: Each method will perform better than the one it extends. HyperNEAT will outperform NEAT, ES-HyperNEAT will outperform HyperNEAT, and DES-HyperNEAT will outperform ES-HyperNEAT.

Hypothesis 2: The running time of the indirect algorithms will increase compared to the one it extends. Because of the ES-HyperNEAT search and multiple searches in DES-HyperNEAT the algorithms running time will be longer for each extension.

The results of the experiment is shown in Figure 5.7. The figure shows the mean fitness of the champion for each generation in all 20 runs. NEAT performed best of the methods, and the fitness at the end was significantly ($p < 0.05$) higher compared to the other methods. DES-HyperNEAT performed better than HyperNEAT and ES-HyperNEAT with LEO, having a significantly ($p < 0.05$) higher fitness after 350 generations. HyperNEAT and ES-HyperNEAT with LEO had a very similar performance. Figure 5.8 show the number of maze completion over the 20 runs. Neither HyperNEAT nor ES-HyperNEAT was able to complete the maze in any of

the 20 runs. NEAT completed the maze in 8 out of 20, while DES-HyperNEAT completed 3 out of 20. This shows Hypothesis 1 is incorrect because the most simple method NEAT performed best of the methods. This could be the case because the complexity needed to solve the maze navigation domain is not requiring an indirect encoding.

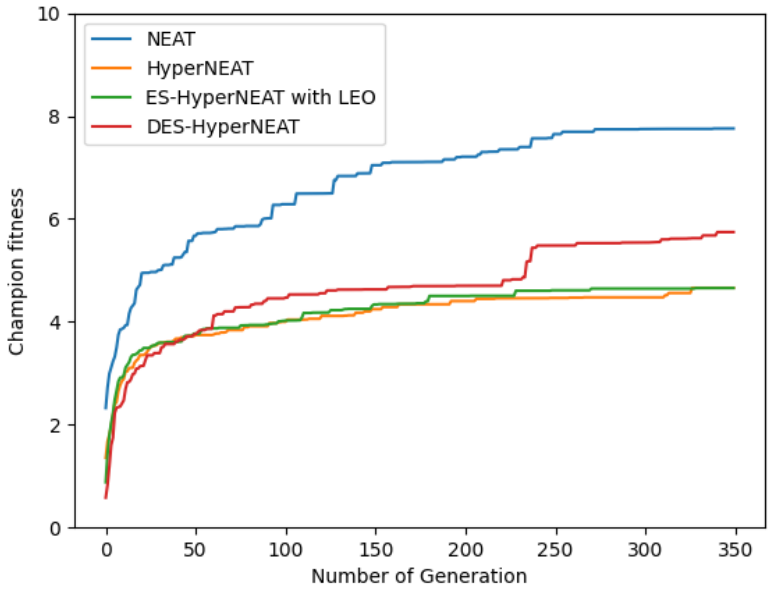


Figure 5.7: The mean champion fitness for each generation over 20 runs.

The mean running time for the NEAT algorithm was 491 seconds, which was significantly faster than the HyperNEAT, which had a mean running time of 783 seconds. HyperNEAT was then significantly faster than ES-HyperNEAT (1184 seconds) with LEO which was significantly faster than DES-HyperNEAT (1483 seconds). Hypothesis 2 was therefore correct.

The conclusion is that DES-HyperNEAT performs significantly ($p < 0.05$) better than both ES-HyperNEAT and HyperNEAT in the maze navigation domain. It does, however, perform significantly ($p < 0.05$) worse than NEAT. It remains to be seen how DES-HyperNEAT performs in a reinforcement learning problem with a higher number of inputs and outputs as this is a domain HyperNEAT methods previously have outperformed NEAT.

5.8 Experiment 4: Comparing against Proximal Policy Optimization

Experiment 4 compares DES-HyperNEAT to the state of the art Proximal Policy Optimization (PPO). The experimental parameters are shown in Table 5.10. The PPO implementation used was created by Raffin et al. [2021]. The implementation has the ability to run on a GPU, therefore, the algorithm could be parallelized, and the efficiency could be improved. The efficiency is therefore not compared in the experiments. As PPO does not utilize the concepts of generations as DES-HyperNEAT, the number of evaluations is used as a stop criterion to get a fair comparison. The number of evaluations, 12000, equals 120 generations of DES-HyperNEAT.

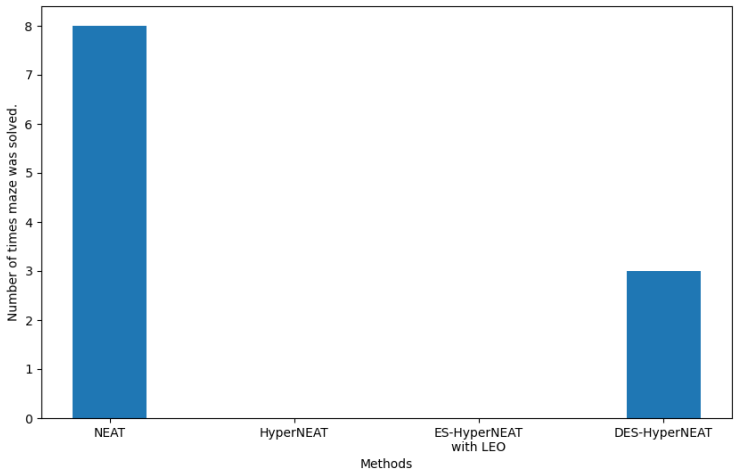


Figure 5.8: The number of times the methods was able to solve the maze in 350 generations out of 20.

	Experiment 4
methods	[DES-HyperNEAT, PPO]
domain	Hard maze
stop criterion	12000 maze evaluations
repeats	20

Table 5.10: Experimental parameters for experiment 4.

The reward for the agent in PPO is calculated by the movement towards the next stepping stone. If the agent moves in the wrong direction, it will be penalized. This way, the agent can not abuse the reward system by going back and forth. If a stepping is passed, a bigger reward is given, and a maximum reward is given when the agent reach the goal. The agent uses an ANN with two hidden layers with 64 nodes in each layer for the action policy. This was the default from the library used for the input type of the agent.

Figure 5.9 shows the results of Experiment 4, the fitness of the resulting ANNs in the maze domain has been used for comparison. PPO performed significantly ($p < 0.05$) better than DES-HyperNEAT, and the fitness for 75% of the runs was over the maximal fitness achieved by DES-HyperNEAT. The conclusion is that DES-HyperNEAT is still far behind a state of the art specific reinforcement learning algorithm for the hard maze navigation domain.

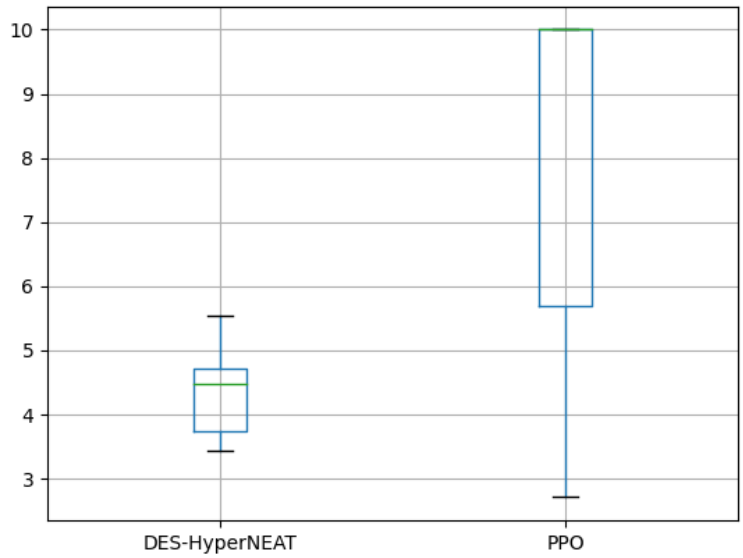


Figure 5.9: Boxplot of DES-HyperNEAT and PPO based on fitness in the maze domain, results after 12000 evaluations.

Chapter 6

Conclusion

In this chapter, the goal and research questions are revisited in section 6.1. While section 6.2 and section 6.3 explains the contribution to the field and potential direction for future work.

6.1 Evaluation and Discussion

This section evaluates the four research questions explored to accomplish the goal. The research questions will be presented, followed by evaluations and further discussion.

Research question 1 *How should bias be applied to DES-HyperNEAT to positively affect performance?*

Bias was added to DES-HyperNEAT to increase the performance of the assembled ANNs. Two ways of applying bias to the ANN were proposed. The first way, multiple CPPN bias, was accomplished by adding an extra output node to each CPPN in the layouts. This node was used to assign the bias to a node upon discovery. The other way of adding bias to the ANN was accomplished by having a dedicated bias CPPN for each individual in the population used to assign bias to all nodes in the assembled ANN. The latter variant was called single CPPN bias.

Single CPPN bias had the best performance out of the two proposed variants compared to the DES-HyperNEAT with no bias. This was surprising as the multiple CPPN bias variant assigned more specific biases during discovery, while single CPPN bias was thought of as a more efficient method by having less complex CPPNs to optimize. The increased search space might have been the factor that made multiple CPPN bias less advantageous. The evolution might, therefore, struggle with optimizing the CPPNs for both finding great weights and biases. Single CPPN bias had, however, not significantly worse performance nor efficiency than DES-HyperNEAT with no bias, but showed no sign of advantages.

Research question 2 *How do Link Expression Output's features benefit multiple substrates in DES-HyperNEAT compared to single substrates in ES-Hyper-NEAT?*

Link Expression Output (LEO) was added to ES-HyperNEAT and DES-HyperNEAT both with and without seeding of geometric concepts. The CPPN in ES-HyperNEAT got an extra LEO node, while in DES-hyperNEAT each CPPN in the layouts got an extra LEO node. When seeding was added, every new CPPN was initialized with the seed.

The performance of ES-HyperNEAT increased slightly with the LEO node added, but there was almost no difference in the variants with or without the seeding. The more surprising part

was that there was no efficiency difference between the ES-HyperNEAT without LEO and ES-HyperNEAT with LEO and without seeding. This proved the Hypothesis that ES-HyperNEAT with LEO requires more time as the CPPN needs to be more complex was incorrect. As the efficiency was significantly worse with the seeding, the variant without seeding would be preferred for solving the maze navigation domain.

DES-HyperNEAT was not able to get any increase in performance with any of the LEO variants as the performance got significantly worse with the extension. The efficiency was also significantly worse with LEO activated. This shows that the modularity in DES-HyperNEAT decreases the need for LEO compared to ES-HyperNEAT. The ability to add new substrates instead of increasing the density inside a substrate seems to mitigate the need for LEO as then all CPPN get more complex.

Research question 3 *How does DES-HyperNEAT perform compared to NEAT, HyperNEAT, and ES-HyperNEAT in a reinforcement learning environment?*

Experiment 3 compared multiple related neuroevolution methods in a reinforcement learning environment that optimizes a controller for an agent in a maze. In the comparison of the direct encoded method, NEAT performed significantly better than all the indirect encoded methods. Within the indirect encoded methods, DES-HyperNEAT performed significantly better than both ES-HyperNEAT with LEO and HyperNEAT, which performed very similarly. The low complexity needed to create a functioning agent controller might be the reason NEAT performed well. As NEAT generally performs well when evolving small networks as the networks gradually get more complex. The gradual complexification of NEAT, therefore, finds an appropriate network size. As this principle is adapted in DES-HyperNEAT to the layouts, the number of substrates is kept at a suitable number, and the performance might therefore be closer to NEAT.

Research question 4 *How does the generic algorithm DES-HyperNEAT perform compared to a tailored state-of-the-art reinforcement learning algorithm?*

The performance of DES-HyperNEAT was compared with Proximal Policy Optimization (PPO), a tailored reinforcement learning algorithm. The performance of PPO was significantly better than DES-HyperNEAT after the same number of evaluations. This shows that the generic indirect encoded DES-HyperNEAT is behind the tailored methods within RL.

Goal *Investigates how DES-HyperNEAT can be extended to solve complex reinforcement learning problems.*

The overall goal has partly been accomplished as DES-HyperNEAT was able to evolve functioning controllers for the maze navigation domain. None of the extensions of adding bias to the assembled ANN showed any advantage in performance. The extension of LEO also performed worse than DES-HyperNEAT without, both with and without seeding. DES-hyperNEAT was, however, the best performing method of the indirect encoded NE methods evaluated and has shown potential in reinforcement learning problems.

6.2 Contributions

The contribution of this thesis has been the extension and comparison of DES-HyperNEAT on reinforcement learning problems. The extension exists of the proposed variants of bias added to the assembled ANN and the proposed way of adding LEO to all CPPNs in DES-HyperNEAT. The

LEO extension has also been added to the ES-HyperNEAT for the maze navigation domain and a comparison of the effect on single and multiple substrates has been conducted. A comparison of related methods has been completed, comparing DES-HyperNEAT with NEAT and HyperNEAT-based methods in addition to a state of the art reinforcement learning algorithm. In addition to the extension and comparison, the implementation of DES-HyperNEAT for RL is a contribution.

The open-source implementation is also a contribution. The final implementation of DES-HyperNEAT for Reinforcement Learning is available as a git repository at Svendsen [2022].

6.3 Future Work

The maze navigation domain has been used to evaluate and compare DES-HyperNEAT in this thesis. The domain has previously been used to evaluate HyperNEAT and ES-HyperNEAT, which DES-HyperNEAT extends. As DES-HyperNEAT performed better in this domain than the other indirect encoded methods, it would be beneficial to compare DES-HyperNEAT in a domain where HyperNEAT methods outperform NEAT. Primarily in domains with a high number of input nodes. As DES-HyperNEAT both previously and in this work has exceeded HyperNEAT and ES-HyperNEAT, it will likely perform well in these domains.

Another interesting research topic would be exploring new ways of adding bias to the assembled ANN as neither the single CPPN bias nor the multiple CPPN bias showed improvement in the experiments. As the extension of LEO showed poor performance, it would be interesting to do extensive research on how the connectivity between and inside substrates in DES-HyperNEAT behaves when the size network heavily increases.

Bibliography

Gillespie, L. E., Gonzalez, G. R., and Schrum, J. (2017). Comparing direct and indirect encodings using both raw and hand-designed features in tetris. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 179–186.

Green, C. (2003). Sharpneat homepage.

Hausknecht, M., Lehman, J., Miikkulainen, R., and Stone, P. (2014). A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Huizinga, J., Clune, J., and Mouret, J.-B. (2014). Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 697–704.

Kassahun, Y. and Sommer, G. (2005). Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *ESANN*, pages 259–266.

Koutnik, J., Gomez, F., and Schmidhuber, J. (2010). Evolving neural networks in compressed weight space. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 619–626.

Lehman, J., Stanley, K. O., et al. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336. Citeseer.

McIntyre, A., Kallada, M., Miguel, C. G., Feher de Silva, C., and Netto, M. L. (2015). neat-python.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2019). Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier.

Montana, D. J., Davis, L., et al. (1989). Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767.

Omelianenko, I. (2019). *Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms*. Packt Publishing.

Pugh, J. K. and Stanley, K. O. (2013). Evolving multimodal controllers with hyperneat. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 735–742.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

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 conference on Genetic and evolutionary computation*, pages 563–570.

Risi, S. and Stanley, K. O. (2011). Enhancing es-hyperneat to evolve more complex regular neural networks. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1539–1546.

Risi, S. and Stanley, K. O. (2012). An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial life*, 18(4):331–363.

Schaffer, J. D., Whitley, D., and Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Sosa, F. A. and Stanley, K. O. (2018). Deep hyperneat: Evolving the size and depth of the substrate. *Tech. Rep.*

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162.

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.

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

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.

Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Lv, J. (2020). Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9):3840–3854.

Svendsen, K. M. (2022). Des-hyperneat for rl. https://github.com/Maagero/DES-HyperNEAT-RL.

Tenstad, A. (2020). Deep evolvable-substrate hyperneat-extending es-hyperneat with multiple substrates in an evolving topology. Master's thesis, NTNU.

Tenstad, A. and Haddow, P. C. (2021). Des-hyperneat: Towards multiple substrate deep anns. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 2195–2202. IEEE.

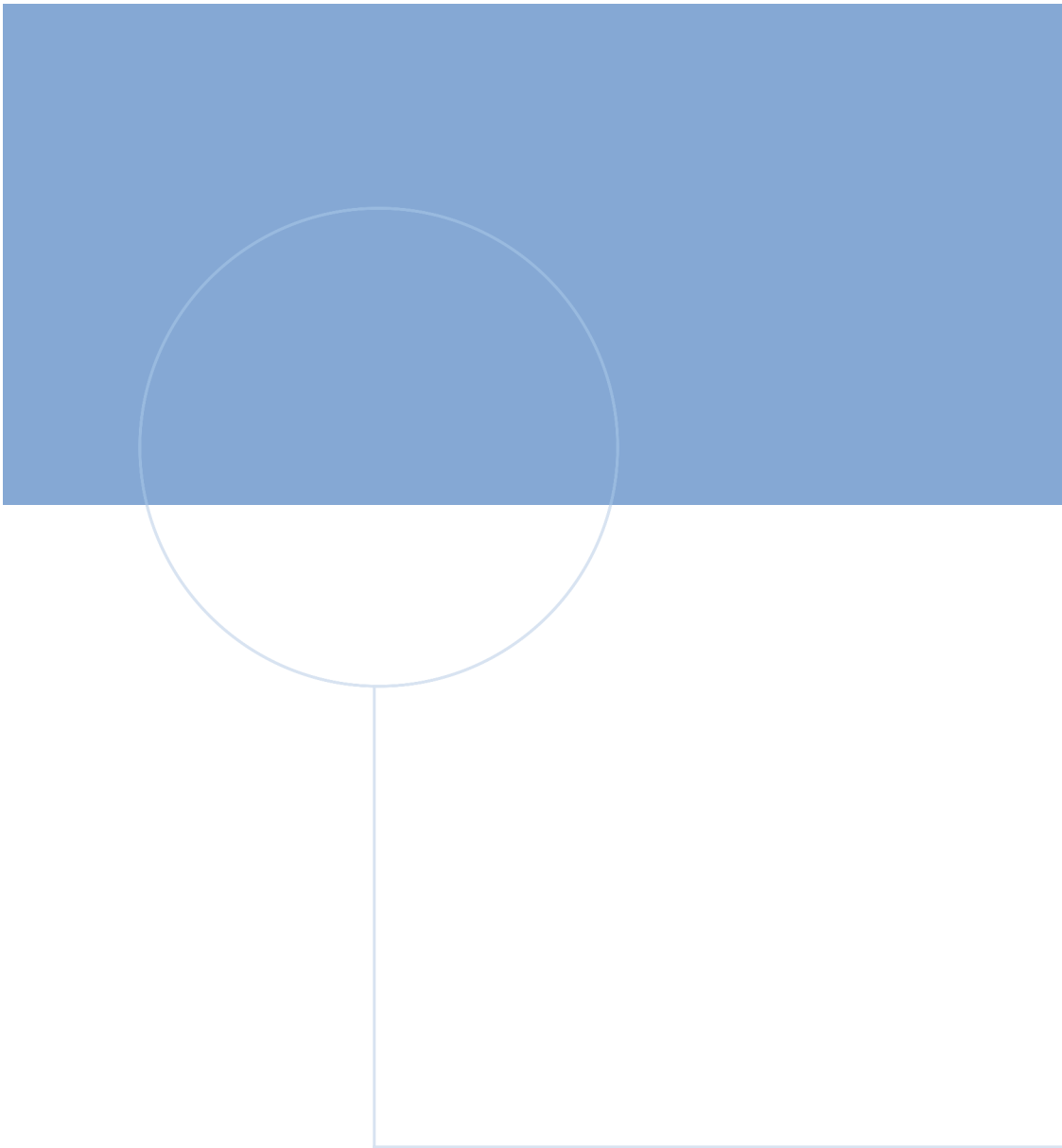
Vanvuchelen, N., Gijsbrechts, J., and Boute, R. (2020). Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry*, 119:103239.

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*, pages 1483–1490.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, S., Zhou, T., and Bilmes, J. (2019). Bias also matters: Bias attribution for deep neural network explanation. In *International Conference on Machine Learning*, pages 6659–6667. PMLR.

Westh et al., A. (2017). Pureples - pure python library for es-hyperneat.



NTNU

Norwegian University of
Science and Technology