Student Thesis

Level: Masters

Developmental Encodings in Neuroevolution - No Free Lunch but a Peak at the Menu is Allowed

Author: 1) Bala Kiran Manthri

2) Kiran Sai Tanneeru

Supervisor: Arend Hintze

Examiner: Moudud Alam

Subject/main field of study: Microdata Analysis

Course code: MI4001

Credits: 30 ECTS

Date of examination: 09-06-2021

Abstract:

NeuroEvolution besides deep learning is considered the most promising method to train and optimize neural networks. Neuroevolution uses genetic algorithms to train the controller of an agent performing various tasks. Traditionally, the controller of an agent will be encoded in a genome which will be directly translated into the neural network of the controller. All weights and the connections will be described by their elements in the genome of the agent. Direct Encoding – states if there is a single change in the genome it directly affects a change in the brain. Over time, different forms of encoding have been developed, such as Indirect and Developmental Encodings. This paper mainly concentrates on Developmental Encoding and how it could improve NeuroEvolution. The No-Free Lunch theorem states that there is no specific optimization method that would outperform any other. This does not mean that the genetic encodings could not outperform other methods on specific neuroevolutionary tasks. However, we do not know what tasks this might be. Thus here a range of different tasks is tested using different encodings. The hope is to find in which task domains developmental encodings perform best.

Keywords: NeuroEvolution, Genetic Algorithms, Direct Encoding, Indirect Encodings, Developmental Encodings, MABE

INDEX

List of Tables:

List of Figures:

1. Introduction

Search algorithms are typically applied to problems that can not be solved otherwise, such as backpropagation/deep learning (Norvig & Intelligence, 2002) or reinforcement learning. The search should be as efficient and thus fast as possible, reducing the computational cost and waiting time. This suggests that one should look for a search algorithm that is optimal for each problem. However, the No Free Lunch Theorem (Schaffer, 1994; Wolpert & Macready, 1995; Wolpert, 1996; Wolpert & Macready, 1997) found that this is an impossible task. No Free Lunch Theorem (NFL) states that when applying different search algorithms on all possible problems, it turns out that on average they all perform the same. While this makes the search for a perfect search algorithm futile, it does not prevent us from finding what kind of search algorithm is optimal for a particular subdomain of problems. After all, while maybe not being able to deploy the perfect one for all problems, it might be good enough to use a specific algorithm optimal on a specific domain, as it has been shown before in neuroevolution (Hintze, Schossau, & Bohm, 2019).

While the NFL theorem applies to search, here we are interested in optimizing artificial neural networks using genetic algorithms (GA) (Goldberg & Deb, 1991). Since GAs belongs in the class of search algorithms, the NFL theorem should apply to them as well, however, a GA is also not a strictly optimal but stochastic search algorithm. In other words, while it is likely that the NFL theorem applies to GA's, it also has not been proven, but we assume as much. Regardless, artificial neural networks (ANNs) are a powerful tool inspired by biological brains (Norvig & Intelligence, 2002). The structure of an ANN is defined as a layer of neurons that are interconnected with each other with randomly assigned weights. The weights in each layer would be described as a matrix of weights and additionally, it has a vector of bias weights. The goal of the ANN is to solve a task (same as the brain) but instead of using biological learning, algorithms such as backpropagation are used to optimize the weights. Backpropagation is a technique that is used to train the weights in the network during the backward path until the performance of the network is satisfying. Deep learning is a part of ANN with multiple layers of hidden nodes between the input and the output layer.

Neuroevolution takes a different approach to optimization than deep learning. In Neuroevolution, a population of solutions, here recurrent neural networks, are optimized by a GA. The GA at every generation evaluates every solution using a fitness function (sometimes called objective function). Then, those solutions that perform better are selected to create the next generation. When doing so, each new solution experiences changes in the form of mutations. This process of mutating typically creates many deleterious or faulty solutions, but occasionally better ones. Since better solutions are being kept, over many generations a GA can find better or more optimal solutions. However, crucial for this investigation are the mutations. Weak or small mutations will slow this process, while strong or large mutations might also not be optimal, as they are doing more harm than good. Consequently, the strength of mutations and what they change ultimately determines the speed of the GA.

In neural networks, the weights in the layer will be changed during the backpropagation but in GA's mutations in the genome will be changing the weights of the matrix. If there is any mutation in the genome the corresponding weight matrix in the Recurrent Neural Network (RNN) will be changed. This is called *direct encoding* because single mutations translate directly into single weight changes. In *indirect encodings*, a single genome will not represent a single weight matrix in the RNN but the relation is more complicated, so if any mutation happens in a genome multiple weights will be changed or impacted with the change in the mutation. Hyperneat is considered the most successful example of Indirect Encoding (Stanley & Miikulainen, 2002). In HyperEncoding the genome specifies the function of a Compositional Pattern-Producing Network (CPPN). This CPPN now is used to specify the weights of each weight matrix. It does so, by taking the x, and y location of the weight as input and performing complex transformations such that a single weight can be derived. Mutations in the genome will now change the Compositional Pattern-Producing Network, and thus have large and complicated effects in the weight matrices specified by the CPPN.

Biological Neural networks are never directly encoded. In natural organisms, genes mostly encode proteins and transcription factors that regulate the expression of other genes. Proteins then control the function of a single cell. Cells divide, communicate, migrate, and form tissues and organs during embryonic development. One of those organs is the neural network of the brain. Technically, the brain is not only growing like most other organs, but it is also constantly adapting (learning) during the lifetime of the organism. Mutations create small changes in proteins, and their effect is translated in an extremely complicated fashion into the resulting phenotype - not at all a direct mapping between mutation and effect.

This kind of developmental encoding is not only indirect but also happens over multiple time steps. Thus, it could be considered its category, even though it is just another maybe more complicated form of an indirect encoding. When trying to extend the analogy to ANN, we would not only need a genome and an indirect translation into weights but also a process that does that translation, like the embryonic developmental process. Such a Developmental Neural Network (DNN) still experiences mutations in the genome, but the effects of those mutations would be translated into weight changes using a developmental process. Hence the process of mapping in the DNN unfolds the information in the genome over multiple time steps rather than in one-off time step as the indirect encoding. Several different developmental processes have been proposed, such as L-systems (Lindenmayer, 2009) or other rule-based methods. Here, a gene regulatory network (GRN) model will be used as the developmental system GRNs are already used to model gene and transcription factor interactions and are thus suitable for our purposes (Hintze, Hiesinger, & Schossau, 2020).

Extending the domain of indirect encodings to include developmental processes is one thing, however, the expectation is, that these more biological systems should also perform better. At the same time, we know from the NFL theorem, that no system will ultimately be better on all tasks. Previous results, when testing the different encodings describes above, also support this idea (Hintze, Hiesinger, & Schossau, 2020). Here, we want to explore which kind of tasks, different encodings thrive, and where direct encodings might be more optimal.

1.2. Background Study:

The previously performed study on the buffet method (Hintze, Schossau, & Bohm, 2019) tested different computational tasks, and how a GA performs when optimizing different neural computational systems. This study already confirmed that the no-free lunch theorem applies to different neural network structures. The tasks used in that study were already implemented in Modular Agent-Based Evolver (MABE). We took the opportunity to test the different encodings on these previously defined environments by selecting only a few substrates that were performing better in their field. However, we need to make adjustments to adapt the MABE framework to our experiment. Overall, they found that the performance of the buffet method was better than others in most of the tasks but still not an ideal one and it is difficult to define good benchmarks.

Those benchmarks, however, are only considering static environments, and not changing fitness landscapes as we find them in nature. Thus, we also test our DNN on a changing fitness landscape as defined in (Schossau & Hintze, 2020). To test if dynamic environments can be searched better using a developmental encoding.

In (Gillespie, Gonzalez, & Schrum, 2017), the authors Compared direct and indirect encodings by setting up an environmental experiment. They passed the raw inputs through NEAT (Stanley & Miikulainen, 2002) which gave good results but once the number of inputs had exceeded a certain limit, it could not handle it very well. So, then they repeated using the HyperNEAT (Stanley, D'Ambrosio, & Gauci, 2009) – which gave better results. When the same experiment was conducted with hand-designed features, NEAT was initially slow, but it caught up to the performance of the HyperNEAT. They concluded that both have their strengths and hybrid structure can be developed for future work.

However, developmental encodings present another alternative. They take into account that natural organisms have a much more complicated translation from genotype into phenotype than indirect encodings provide. Specifically, one that unfolds over time. Different developmental encodings have been presented in the past (Harding & Banzhaf, 2009). In (Hintze, Hiesinger, & Schossau, 2020) presented three new kinds of developmental encodings, which also have been implemented in MABE. In that study, the developmental encodings were further adaptive to their environment. This kind of adaptation was of no concern for this study. Again, we took advantage of this opportunity to reuse code, instead of implementing other previously developed methods.

MABE is a digital tool developed to make computational modeling experiments easy (Bohm & Hintze, 2017). It is similar to that of a vanilla code on which we can make changes according to our requirements. MABE is a framework that contains different modules, like genomes, brains, worlds, optimizers, groups. These modules are interconnected. Before the difficulty of digital evolution was more complex and time taking, so MABE was developed to make it simpler and more effective than its predecessor. However, the task environments were implemented in an older version of MABE, but the brains were implemented in a newer version. Thus, before experiments could be conducted, code had to be migrated and updated from one version into the other. Interestingly, the core design principle of MABE is to create modules that can be reused and recombined with each other. This work is a testament to the success of this design principle.

1.3. Objectives:

In this paper, we have explored different types of encoding over different possible tasks by developing an agent. Through this, we wanted to understand does the structure of encoding matters. And we also wanted to know does the developmental encoding outperforms direct encoding in any environment. The task can be achieved by:

1. By testing all the brains on all the static environments
2. By testing all the brains in a dynamic environment - but the environment differs from each other in the speed at which they change.

We wanted to look into a certain category of encoding and understand which type of encoding performs better in different conditions. As we are looking at a subset of brains and environments and understand which brain is optimal in which environment. The real question can be are there any pairs that are better than other combinations. This confirms that it's not always one brain or environment which performs better all the time. Through this, we can confirm that the NFL still applies to those certain sets of brains and environments.

2. Methodology

MABE(Modular Agent-Based Evolver) (Bohm & Hintze, 2017) is a digital tool that allows us to implement the Digital Evolution on it. It has basic elements, and the users can modify it according to their requirements. It was first introduced in 2016, and from then on it was modified according to the requirements. Basically, in terms of a simple understanding, it is similar to a vanilla code. Vanilla code can be understood as a basic code for a particular module, this vanilla code can be modified to gain the desired outcome. Similarly, MABE does not follow a particular working structure, its main purpose is to provide a general-purpose digital tool that can be based on many solutions. MABE has a framework that consists of several modules like Brains, Genomes, Optimizers, Worlds, and Archivist.

2.1 MABE Design:

The main goal of MABE is to build software that helps the user to create a custom version by decreasing the build time and cost. When MABE is installed the default version is configured with a typical MABE case. The example contains Organisms that have a Circular Genome that encodes the Markov Brain. A GA Optimizer (Method to generate a new population using roulette wheel selection) selects the highly scored optimizer to form a new population. Archivists are used to saving and tracking the data. Understanding a MABE is very easy and quick. MABE generates the configuration files automatically with the embedded documentation. These files help the user to alter the MABE behavior by changing the parameters such as the modules like brains, genomes, environments, or the population size or run duration.

Since various elements, worlds specifically, needed to be ported from an older version of MABE to a newer one, as well as they needed to be integrated into their new context, the structure of MABE is laid out in the next paragraphs.

2.2 Modules in MABE:

2.2.1 Genomes:

Genome is the genetic structure of an organism with four nucleotides – A, C, G, and T. These will play a key role during evolution. Mutations will apply small changes to the organism's performance and are thus the elements on which evolution progresses. But here in our study, we are not restricted to these four types of nucleotides, our genome is a list of continuous or discrete values necessary to encode the brains investigated here. Genomes experience point mutations (that is single site changes), duplications, and deletion. Point mutations happen with a likelihood of 0.005 per site, deletions, and gene duplications of 128 to 512 nucleotides (determined by a uniform random distribution) happen with a probability of 0.1 per replication of the genome. Genomes are used by Brains and are used to encode how each individual type of brain is encoded.

2.2.2 Brains:

The brain is the key organ for most organisms. The brain takes the inputs from various sources, then processes the information accordingly and gives the output. In the same way, Brains in MABE act as the data processing units which receive inputs and deliver outputs. In this study, we have considered five brains. The difference between each brain can be described as how they convert the inputs to outputs using the internal algorithms. Our brains are built by using the genome structure by encoding it directly or indirectly. Each brain has its computational algorithm. These brains do not have any information regarding the genomic structure, they will just take the information, process it, and give the output. Different types of brains in MABE are:

Direct Encoding: In direct encoding, the genome is directly connected to the weight matrix in the RNN. For a single change in the genome, a single weight matrix will be changing in the RNN.

Developmental GRN Encoding: In Developmental GRN encoding we have a genome that is given as a weight matrix to the Gene Regulatory Network (GRN). GRN performs complicated mathematical operations for a certain specific number of steps resulting in a gene expression vector. The expression vector is sequentially translated as a weight matrix which is then fed to the RNN. As the expression is indirectly connected to the weight matrix, any single change in the genome will be resulting in multiple changes in the weight matrix of RNN. As the values from the genome given to the GRN, it can make an inverse impact on the resulting values.

Indirect Hyper Encoding using CPPN: It is a type of indirect encoding, where the values in the genome are translated to CPPN. Then the CPPN is used to populate the weight matrix of the RNN.

Compositional Pattern-Producing Network (CPPN): The CPPN continuously receives two values from the genome and results in a single output value. It is distributed over 16hidden nodes, each having two input nodes and one output node.

Indirect Hyper Developmental Encoding (CPPN-GRN): It is a type of hybrid indirect encoding where we have a genome that is given to the CPPN and the CPPN populates the weight matrix and it is passed to the GRN and the GRN runs for some specific number of time resulting in a gene expression vector which is given as weights to the RNN.

Developmental Hyper Encoding (GRN-CPPN): The genome is translated as the weight matrix to the GRN and GRN's run for some specific number of steps resulting in a gene expression vector which is used to define a CPPN and then CPPN's output values are given as weights to the RNN.

2.2.3 Optimizers:

Optimizers use the information world generates about the performance of the agents and use this information for creating the next generation. They are in essence the "selection" part of evolution. Here specifically, a genetic algorithm was used, which selects genomes to be copied into the next generation proportionally to the performance of their associated agent. This method is allied roulette wheel selection. Other options are available, but since we are not interested in other forms of selection they are not any concern.

2.2.4 Archivist:

Archivists determine what kind of data needs to be stored and when it needs to collect data. Archivists generate the files in a CSV format which is human-readable and supported by many tools. Generally, archivists produce two types of files. One is temporal files which have the data record over time, which means it has the data regarding the highest-scoring organism in the population or the average of all organisms in the population. The second file is a record of the genomes along the line of descent (LOD) (Lenski, Ofria, Pennock, & Adami, 2003).

Generally speaking, MABE produces a lot of information, the archivist filters and saves that part of the data the researchers are interested in.

2.2.5 Worlds:

Worlds are simple environments, where we test the capability of the organism which is controlled by a brain. Each world's functionality is different from the other. We test our five brains with each of the following worlds and try to find which brain is performing the best in which environment. For this study, we choose 9 Static worlds and one dynamic changing world. The static worlds come from a previous study about the no-free lunch theorem and thus created these as benchmarks (Hintze, Schossau, & Bohm, 2019). The changing environment was used before to explore the effects of dynamically changing environments on the complexity of the brain (Hintze, Hiesinger, & Schossau, 2020). They are:

XOR world: In this world, a Classic Xor operation is performed (James & Tucker, 2004). Where the brain is given the set of logical gate inputs to the agent and expects the output. The performance of the agent is evaluated by comparing the original output with the output of the agent. The performance is noted down as the score of the agent for that particular generation.

Symbolic Regressor: It is a type of regression analysis. Where the agent will be shown different functions and the performance of the agent is then determined by the difference between the output of the function and the response given by the agent, which is then summed and squared.

Pendulum World: It is an experiment where we have a cart that can move either left or right (Barto, Sutton, & Anderson, 1983). A beam is mounted on the top of the cart which can freely move on a perpendicular axis. The job of the agent is to move left or right in such a way that it needs to balance the beam perpendicular to the cart to which it has been attached.

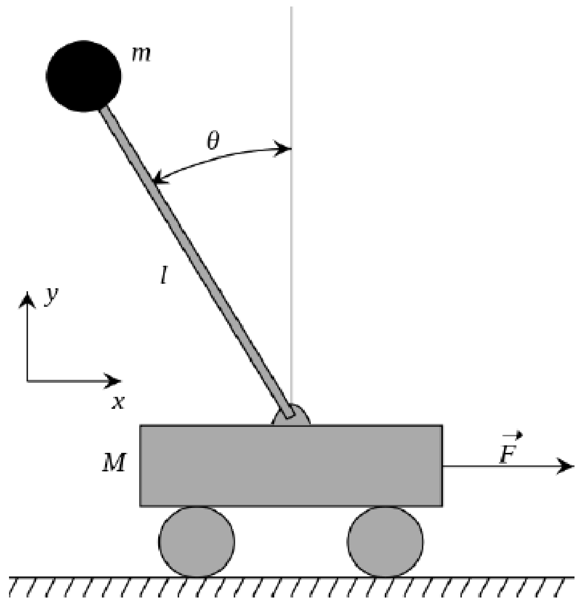

Figure 1: Inverted Pendulum. A weight with mass *m* is mounted on top of a beam with the length *l*. The card is allowed to move forward or backward to balance the pendulum upright. The simulation will be stopped once the angle has reached a certain limit. The cart has a mass of M which can be accelerated with a force F (figure from (Hintze, Schossau, & Bohm, 2019) with permission of the author).

Value Judgment: It is a kind of decision-making experiment. Here the agent will be given two signals and its job is to identify the strongest signal between them (Kvam, Cesario, Schossau, Eisthen, & Hintze, 2015). In our research, a random binary number will decide which signal should be the stronger one resulting in the inputs of the agent. The stronger input will be having a 0.55 probability of being 1 and a 0.45 probability of being 0. Whereas the weaker signal will be having a 0.55 probability of being 0 and a 0.45 probability of being 1. The agent will be provided with 100 input updates and the first 80 updates of results of the agent will be ignored. After the 80 updates, the remaining results from the agent are continuously evaluated and the agent needs to make a decision of which signal is stronger. Then the overall performance of the agent is calculated.

Simple Berry: In this, the agent will analyze the past information and optimize future decisions (Bohm & Hintze, 2017). Let's consider a square block surrounded by a wall and filled with two types of berries red or blue. The agent can move forward, left or right, or consume the food present in the current block. If the agent consumes the food in the current block and moves forward the previous block will be filled with red berry or blueberry randomly. The agent can sense the food present in the current block, i.e blue or red, or empty. Consumption of each berry will be rewarded to the agent, however, if the consumed food differs from the previous food the agent will be negatively rewarded. So the agent needs to consume the food in a way so that he will be having the highest score for the generation.

Block-Catch: Here the agent needs to catch or avoid the blocks which are coming towards them (Beer & others, 1996). Consider a game where a bar is placed at the bottom and one box is falling at a time. But only some of them have rewards and others do not. So the objective of our bar is to catch only boxes with food in it and leave the empty ones.

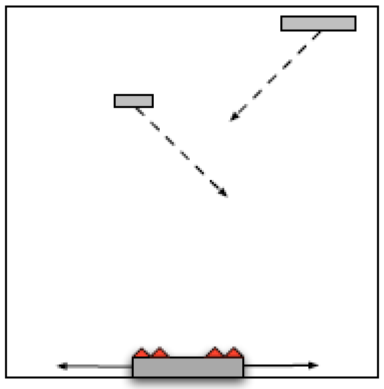

Figure 2: Block Catching. In this experiment, certain large or small blocks will be falling from a certain height and the agent in the bottom can move right or left to catch those blocks. Certain blocks will have a positive reward for catching them and certain blocks will have negative rewards (figure from (Hintze, Schossau, & Bohm, 2019) with permission of the author).

Edlund Maze: In this task, the agent needs to navigate through a path with long walls and a single door as an exit (Edlund, et al., 2011). When passing through the walls, the agent will be receiving a signal, if signal 1 indicates that the next door is towards the right of the agent and no signal indicates the next door is not towards his right. The agent can perform only three actions: Move forward, Move Left or Move Right. The agent needs to remember the previous signal until he finds the next door in the wall.

Path Association: In this task, the agent must follow a safe path through a poisonous environment utilizing the directions like moving forward, left, or right sent by the signal (Grabowski, Bryson, Dyer, Ofria, & Pennock, 2010). Information about the end of the path will not be provided to the agent. The agent can sense the information of its current location: Either the agent is on the path or in the poison.

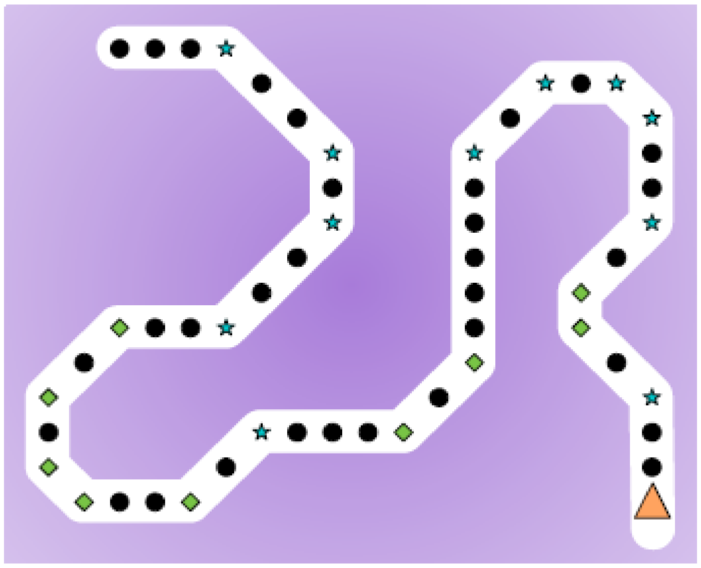

Figure 3: Path Association. The agent (orange triangle) needs to navigate in the path (white) on which there are certain signals that the agent can only sense when he is standing on the signal. The black circle is constant which indicates that the path continues in the straight line. Green diamonds and blue stars indicate that the path will turn left or right, with the randomized meaning of the signal at the start of every experiment, which means that the agent needs to explore what the signal means and then exploit the information. Purple path defines the poisoned environment which has the negative consequences being into the path. (Figure from (Hintze, Schossau, & Bohm, 2019), with permission of the author)

Noisy Foraging: In this task, the agent needs to find food placed on a random grid, once the food is collected the agent needs to return to the home position to increase fitness. For this, the home position is marked in the grid and the agent does not know the position of the food, so the agent has to find these positions, mark them, take the food and return to the home position. At each collection of the food, the placement of the next food is moved farther away from home. The agents have eight detectors each covering a 45 deg arc which is used to detect the food and the home location. The agent also has some additional sensors which inform the agent when it reaches the home or where he can find the next food location.

Dynamic Changing Environment: In this task, the agent will be provided with 4 different kinds of food. The combination of each food will be positively or negatively rewarded to the agent. The agent needs to learn to serve in the specific environment by eating the right combination of foods. As this is a changing environment the agent needs to be adapted to the environmental changes. The environment changes happen from a fast-changing to a very slow-changing and no change (0.0001, 0.001, 0.01, 0.1, and 0). The performance of the agent is calculated by the amount of food eaten multiplied by the reward of the combination of the food added to the overall food eaten.

2.3 Experimental Design:

All the environments were present in the old MABE and the brains were present in the new MABE. During our first phase, we need to adapt our environments from the old MABE with the new MABE. After bringing all the environments and brains compatible with the new MABE. Then for each combination of brains, we need to set up our environments. And we choose the roulette wheel section as the optimizer. To complete our run fast we have given our combination of brains and environments to MABE on a high-performance computer. The setup is described in such a way that each experiment should run over 100 replicates with 10000 generations each for the static environment. After the run is completed, we will be given the output as the LOD files. An example of a file is LOD_5_2_20. Where 5 represents the Environment name, 2 represents the Brain name and 20 represents the replicate. Each file will have a Generation number, ID, and Score. For the Dynamic Changing Environment, the experiment should run over 30 replicates with 200,000 generations each. After the successful run, the output file will be given as LOD_2_0.01_20. Where 2 represents the Brain and 0.01 describes the speed of the changing environment and 20 represents the replicate. Each file will have to have Generation Number, ID, Score, hits, bites, a combination of 4*4 with count and the reward, and a Percent Rank.

During our Analysis phase, for the static environment, we tried to exclude all the replicates which run less than 5000 generations. And for the replicates from 5000 generations to the 9900 generations, we filled the recordings till the 10000 generations with the last recorded value. Sometimes compute nodes on the high-performance computing cluster (UPPMAX) fail or time out, then runs are accidentally too short and need to be sorted out. The LOD tracks the organisms until the point of convergence, which is on average ~2 times the population size before the final generation. All organisms after the point of convergence are not reported, but the organism at the point of convergence can be considered the representative solution at the end of evolution. Thus, the missing data points until generation 10000 can be filled with copies of that organism.

Recordings from all the files based on the brain and environments concatenated into their respective data frames using python (pandas). The respective data frames have been grouped to calculate all necessary variables for the analysis from the recordings across all the data frames.

For a Dynamic environment, we need to understand how the agent has been adapted to the changing food and how better it has been performing. This is somewhat of a problem. In a static environment, the maximal performance is known, in the dynamically changing one, we need to constantly recompute which performance is optimal. The measured performance of the agent can then be understood relative to the optimal performance, here as a Percent Rank.

To calculate the percent rank at every generation, we need to generate all possible strategies, evaluate their score, and rank the actual score of the evolving agent accordingly. In this environment, the sequence of food cubes eaten determines the performance of the agent. Thus we generate all possible sequences of 8 food cubes eaten. The matrix defining how rewarding each cube is was constructed from the record data. Also, the count of each food the agent collected was stored in a similar 4*4 matrix. All possible strategies were then enumerated and their score computed using the previously defined matrices. Agents often did not eat exactly 8 food codes but more or less, requiring us to normalize the generated hypothetical performances to the actual amount of food eaten. The comparison with those numbers and the performance of the evolved agent defines the percent rank of the agent.

Implement in python, we have replicated a 4*4 matrix of food consumed with iter tools product with a repeat of 8. Then every point in the matrix is divided by the sum of the matrix and stored in a list. The reward of the food from each generation is read from the data frame and stored in another list. Then the two lists are multiplied and then summed over to form a list of all possible best scores. The original score of the agent is taken from the data frame and stored in a list. Now, each possible score is compared with the original score of the agent, this is done by finding the number of possible scores which are greater than the score of the agent summed and divided by the length of the possible score. The solution is stored in a list named Rank. The list rank describes how optimal the agent has performed for the generation considering how better he has performed in the generation.

As far as we know, this kind of performance assessment for dynamically changing environments has not been applied before.

2.4 DATA

The data is stored from the output result using the MABE framework. The MABE framework has been set up to store different types of data from static and dynamic environments. After the setup is completed in the MABE it will be testing all environments and all brain combinations but also testing each combination 100 times independently of each other to allow for better results.

MABE files use a naming scheme so that conditions and replicates can be easily identified. The data file from the static environment will be presented as LOD (Line of Decent) followed by Environment number and Brain Number. The environment number describes the environment name and the brain number describes the direct, indirect, or developmental brain details. Each combination of environment and brain has 100 replicates and each replicate will have 10000 generations in them. As a sample LOD will be named as LOD_5_2_20 where 5 represents the environment name, 2 represents the brain name and 20 is the replicate.

The data from the dynamic environment is stored differently. In the dynamic environment, the score for the agent will be given based on the food it has consumed over the environment. The food matrix in the dynamic environment is described as a 4*4 matrix and the score will be calculated by the count of the food consumed * reward of the food by the sum of overall food consumed.

Variables we have in our static environment dataset files are as follows:

Generation Number	The generation number of the agent it is performing
ID	The generation ID of the agent it is performing
Score	The performance of the agent in the generation

Table 1: Variable in Static Environment

Variables we have in our dynamic environment dataset files are as follows:

Generation Number	The generation number of the agent it is performing.
ID	The generation ID of the agent it is performing
Score	The performance of the agent in the generation
Hits	The number of hits the agent has taken
Bites	The number of bites the agent has taken
0_0, 0_1,0_2 …. 3_3	It is the 4*4 food matrix. Collects the sum of each food consumed by the agent
Reward 0_0... Reward 3_3	It is the 4*4 reward matrix. It presents the reward of the food over the generation
Percent Rank	Collects the computed percent rank

Table 2: Variable in Dynamic Environment

3. Results:

Our main goal in this research is to observe that the encoding matters and if the developmental encoding outperforms the direct encoding. However, the answer to this question could also depend on environments being static or dynamic. Therefore, we have first prepared MABE to run on each combination of brains and the 9 static environments with 100 individual replicates over 10000 generations each. We expected either of the two following outcomes: Either one brain turns out to be always the most adaptable on all tasks, or, supported by the NFL theorem, we find that certain combinations of environments and encodings evolve the best, and the kind of match is different from the environment to the environment or from the brain to brain. We found that direct encoding performed better or equally well than other encodings, on all tested static environments (see Figure 1). Some of the developmental encodings failed to find the exact solution in the environment at all.

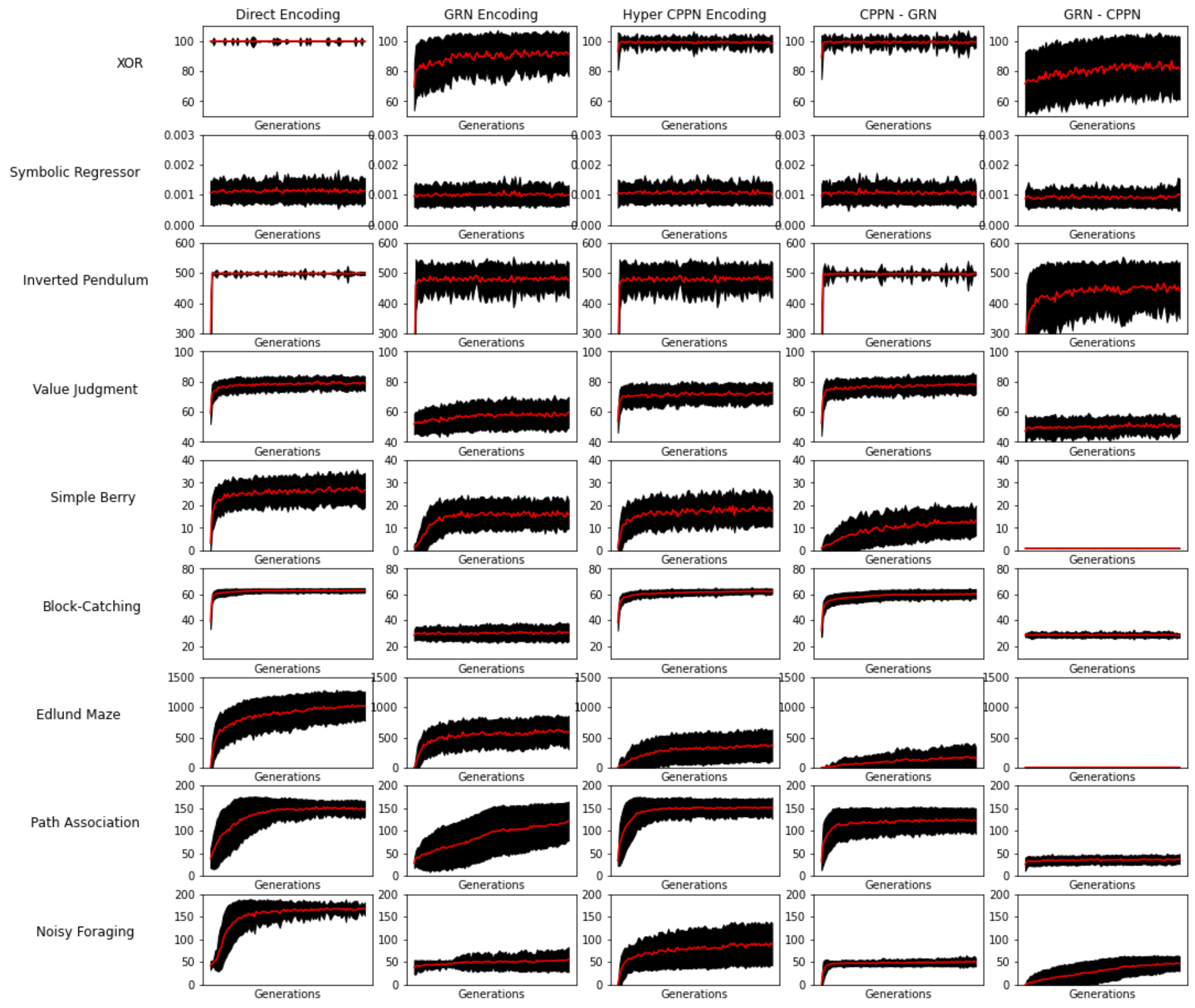

Figure 4: "Performance of Brains over Static Environment", the performance of all the five brains (Direct, developmental GRN, Hyper CPPN, CPPN-GRN, GRN-CPPN) in different static environments has been present, on each figure X-axis is generations, Y-axis is the performance of the brain. The red line indicates the average performance derived from 100 replicate runs of the agent

in the combination of brain and environment and the black bond is the standard error defining upper and lower bounds.

Figure 4 describes the performance of the brains in different static environments. On each figure, the X-axis is generations, and the Y-axis is the performance of the brain. The red line in each figure represents the average performance of the brain over the generations and the black line describes the standard deviation of the performance with the upper limit and the lower limit. From all environments present in the figure we can see that direct encoding was performing good compared to some developmental encoding brains. In some environments, the developmental encoding brains failed to find the solution in the environment. Developmental brains were performing equally to the direct brains in some environments. Noticeably, Hyper CPPN Encoding performed better than the direct encoding (when compared with the average performance) during the first phase of the path association world but the direct encoding performed equal to the CPPN encoding during the second half.

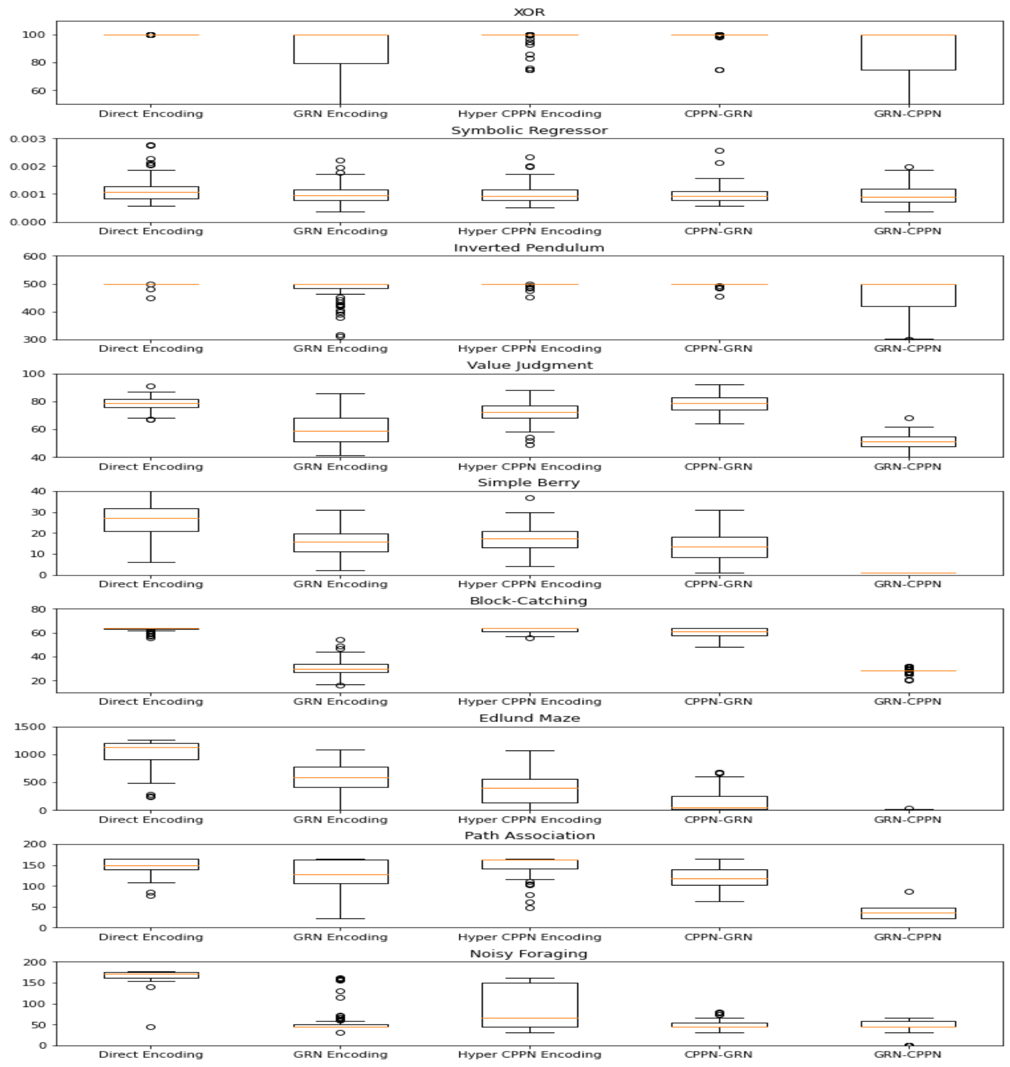

Figure 5: "Boxplot of the Brains over Static Environment", the above figure describes the performance of the brains in different static environments in a boxplot. On each figure the X-axis

defines the different kinds of brains, on the Y-axis defines the performance of the brains in the specific environment.

Figure 5 describes the performance of different brains over the static environment using a boxplot, showing the mean (organs lines), the 25% quartiles (as boxes), and outliers (as circles). On each figure, the X-axis describes the box plot of different brains, and the Y-axis describes the performance of the brain in a different environment. The spread of the points determines how each brain is performing. As an example, when we look into the Block-Catching environment the spread of the points for the direct brain is very less and it stated above when compared to other brains. Overall, direct encoding was better than all the brains in different encodings, but developmental encodings were performing equally in most environments. Also, we can see that the hyper CPPN in the Path Association world was performing better compared to the direct encoding as the median of the hyper CPPN is more than that of the direct encoding. In most cases considered, the respective means of one condition were already outside of the confidence intervals of the other conditions, making a statistical verification superfluous.

From the results of the static environment, we can say that the Developmental GRN and GRN-Hyper were performing poorly when compared with the direct or hyper encoding. Hyper-GRN also performs consistently poorly in some static environments tested above.

However, natural organisms, which are the inspiration for developmental encodings, rarely evolve in static environments, but dynamic ones. While hard to test on natural organisms, examples are showing how changing environments can drive evolution (Steinberg & Ostermeier, 2016). Consequently, for the dynamic changing environment we have prepared our brains to run on different changing environments with the speed of 0.0001, 0.001, 0.01, 0.1, and 0 and having 30 individual replicates with 200000 generations each. In the dynamic changing environment, the developmental brains (GRN) have performed better than the direct brain in all the environments (see below figure 6) followed by the GRN-CPPN and CPPN-GRN encoded brains.

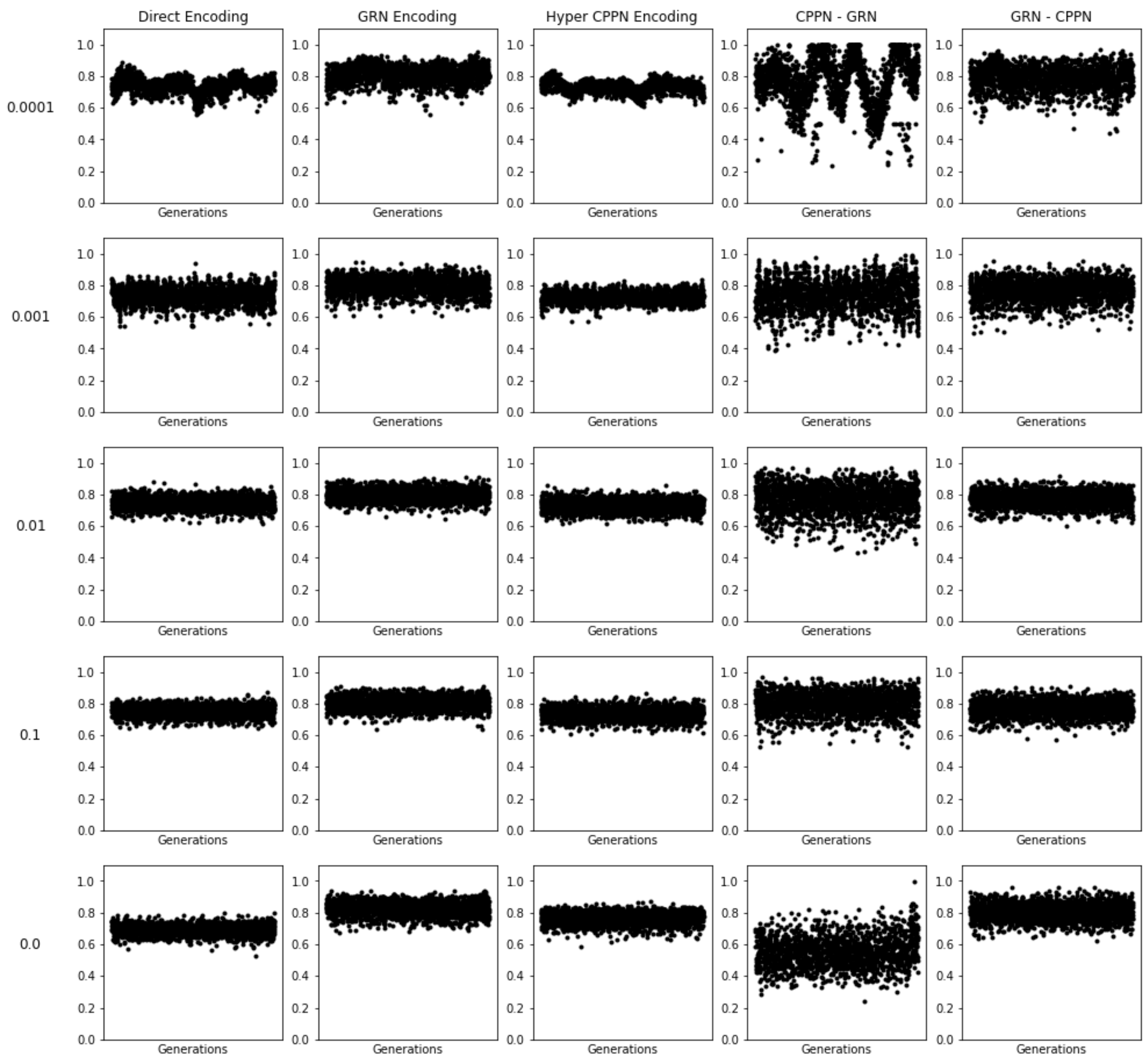

Figure 6: "Percent Rank of Brains in Dynamic Changing Environment", the performance of all the five brains (Direct, developmental GRN, Hyper CPPN, CPPN-GRN, GRN-CPPN) in the different dynamic environment has been present, on each figure X-axis is generations, Y-axis is Percent Rank of the brain. Each point in the combination defines the percent rank which defines how optimal the brain has performed in the specific environment.

Figure 6 describes the percent rank of different brains in a dynamic environment. On each figure, the X-axis is generations, and the Y-axis is the percent rank. Each point in the figure describes the percent rank. The percent rank describes how optimal the brain has performed in the respective generation. Developmental brains were performing better than direct encoding in a dynamic environment. And in the slow environment, 0.0001 GRN Encoding, CPPP-GRN, and GRN-CPPN were performing better than the direct and hyper encoding and in some cases, it has perfectly adapted to the environment by reaching the best optimal performance in the environment compared to the other brains in the environment. However, GRNs perform well, already in the static version (speed=0.0) of this environment, whereas the CPPN-GRN variant does not. Instead, it,s performance improves with increasing speeds, making it the candidate to potentially perform best in dynamic environments, as opposed to the GRN version which seems to generally thrive in this environment.

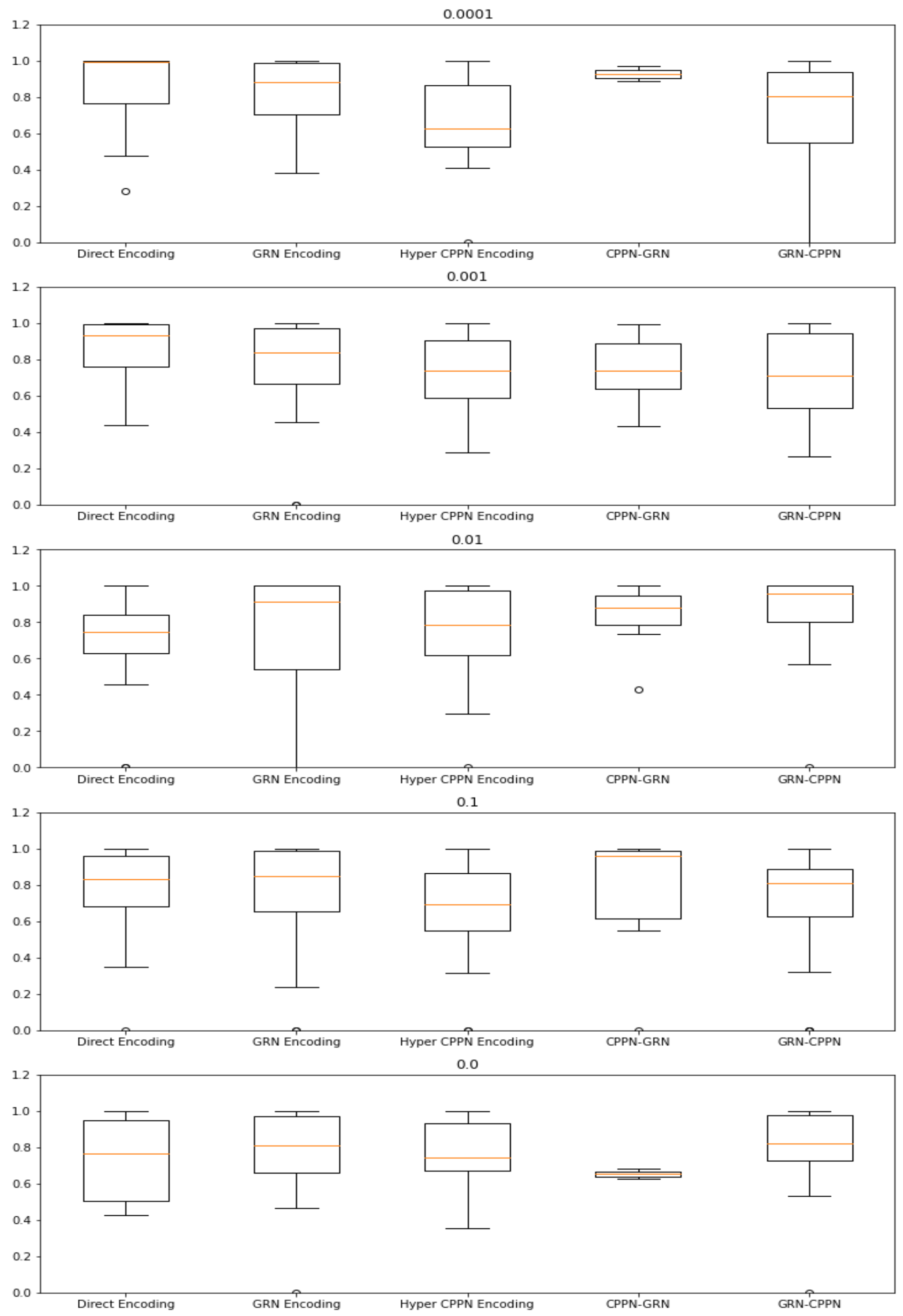

Figure 7: "Boxplot of the Brains over Dynamic Changing Environment", the above figure describes the percent rank of the brains in different Dynamic changing environments in a boxplot. On each figure the X-axis defines the different kinds of brains, on the Y-axis defines the percent rank of the brains in the environment.

Figure 7 describes the percent rank of different brains in a dynamic environment using a boxplot. On each figure, the X-axis defines different kinds of brains and on the Y-axis is the percent rank of the brains. From the figure, we can see that the developmental brains were performing better than the direct brain in all the different dynamic changing environments.

Based on the results from the Dynamic environment, we can say that the developmental brains which failed to find the solution in the static environment are performing better than the direct and hyper brains in the changing environment. It shows that the developmental brains are more suitable for the changing environment and who adapt themself quickly to the changing environment. However, the signal is very weak. The respective means are well within the confidence intervals or even within the 25% quartiles.

Still, hypothesis testing using the Kolmogorov-Smirnov test has been tried across the data points from different environments. Specifically, the Kolmogorov-Smirnov test for two samples, which should reveal if two sets of values were drawn from different distributions, was applied. Unfortunately, p-values were not significant (results larger than 0.1). At the same time, running more replicates in the future could confirm the differences suggested by our results.

4. Discussion:

We have considered an experiment comparing different kinds of brains and understanding does the structure of encoding matter. Based on our results, we can say the direct brains were performing well in static environments where some developmental encoding brains were unable to find solutions for the problem in some environments. But in the dynamic changing environment the developmental encoding brains were performing better than the direct brains. To understand how the brains in the dynamic changing environment we have created a new method called Percent Rank. Percent Rank calculates how optimal the brain has performed in the changing environment which describes how the brains have been adapted to the environment. The continuous computation of percent rank, to our knowledge, has not been done to assess the relative performance of agents in dynamically changing environments, this method might prove useful in the future.

Since we have limited our study with a limited set of brains and environments, and the replicates from the dynamic environment were small. We don't claim completeness in the results as one can use different kinds of brains and environments. But we found evidence that certain encoding performs better than others in a given circumstance. We have completed our experiment using the MABE framework. We always need to check how reliable our code is. There might be bugs? Well, in this case, all modules have been previously tested and published, also, various MABE modules have been tested by many other researchers. So we think there is a high degree of reliability, even though we can not perfectly exclude bugs.

In the future, we would like to test different combinations of brains in different environments with larger generations and replicates which would be great for a more detailed study. This helps to understand how the different combination brains would perform in a larger set of environments with more generations. In the dynamic changing environment the speed of the environment determines which encoding works the best thus DNN would be best suitable for the dynamic changing environment. This may be because different mutations affect the sizes and the mean difference from the distance. Thus we hypothesize that the speed of change in the environment determines the optimality of the mutational distance.

All of this happened using GA as the optimizer. Observing that deep learning very often is preferred in training classifiers, but tasks like the one used here, are more in the domain of controlling robots, autonomous vehicles, or non-classification tasks, the GA is often the only method to solve these problems. Because deep-learned NN does not have encoding in the first place or is always directly encoded, these results do not translate to that domain.

5. Conclusion

In this work, we are trying to understand whether developmental brains outperform direct brains in different environments. We have completed our experiment by preparing an agent using the MABE framework to test our brains in different environments.

Based on our comparison, the direct encoded brain was performing slightly better than the developmental brains in the static environments but in the dynamic changing environment the developmental brains were performing better than the direct brains and the developmental brains were fast in adapting to the changing environment. Some developmental brains which failed to find the solution in the static environment were performing better than those of the direct brains in the dynamic changing environment. However, while suggestive, the results from the tests run using dynamically changing environments have weak or no statistical support. More replicate experiments need to be run in the future (9 days runtime for the whole suite of 30 replicate experiments in dynamically changing environments on the UPPMAX high-performance computer).

In conclusion, we found the no free lunch theorem to still apply, no single encoding performs well across all problem domains, even though we could identify specific combinations of encodings that fair better in some problem domains. We also found some evidence that developmental encodings indeed perform better in dynamically changing environments for these particular sets of brains and environments, but more experiments are needed for a different set of brains and environments. However, we don't claim completeness in the results as one can use different kinds of brains and environments. In practice, when one fails to optimize a neural network by using a genetic algorithm, it is well worth trying different encodings. Specifically, if the task is dynamically changing, developmental encodings might promise better perform

References

Barto, A., Sutton, R., & Anderson, C. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics - V*, 834--846.

Beer, R., & others. (1996). Toward the evolution of dynamical neural networks for minimally cognitive behavior. *From animals to animats - IV*, 421--429.

Bohm, C., & Hintze, A. (2017). MABE (modular agent based evolver): A framework for digital evolution research. 76--83.

Edlund, J., Chaumont, N., Hintze, A., Koch, C., Tononi, G., & Adami, C. (2011). Integrated information increases with fitness in the evolution of animats. *PLoS Comput Biol*, e1002236.

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

Goldberg, D., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms-I* (pp. 69--93). Elsevier.

Grabowski, L., Bryson, D., Dyer, F., Ofria, C., & Pennock, R. (2010). Early Evolution of Memory Usage in Digital Organisms. In *ALIFE* (pp. 224--231). Citeseer.

Harding, S., & Banzhaf, W. (2009). Artificial tendons: early development and application. In *Organic Computing* (pp. 201--219). Springer.

Hintze, A., Hiesinger, P., & Schossau, J. (2020). Developmental neuronal networks as models to study the evolution of biological intelligence. 5.

Hintze, H., Schossau, J., & Bohm, C. (2019). The evolutionary Buffet method. *Genetic Programming Theory and Practice XVI*, 17--36.

James, D., & Tucker, P. (2004). A comparative analysis of simplification and complexification in the evolution of neural network topologies. In *Proc. of Genetic and Evolutionary Computation Conference.*

Kvam, P., Cesario, J., Schossau, J., Eisthen, H., & Hintze, A. (2015). Computational evolution of decision-making strategies. *arXiv preprint arXiv:1509.05646*.

Lenski, R., Ofria, C., Pennock, R., & Adami, C. (2003). The evolutionary origin of complex features. *Nature*, 139--144.

Lindenmayer, D. (2009). *Forest pattern and ecological process: a synthesis of 25 years of research.* CSIRO publishing.

Norvig, P., & Intelligence, S. (2002). *A modern approach.* NJ, USA: Prentice Hall Upper Saddle River.

Schaffer, C. (1994). A conservation law for generalization performance. In *Machine Learning Proceedings 1994* (pp. 259--265). Elsevier.

Schossau, J., & Hintze, A. (2020). Neuroevolution in Dynamically Changing Environments. 744--746.

Stanley, K., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation-X*, 99--127.

Stanley, K., D'Ambrosio, D., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 185--212.

Steinberg, B., & Ostermeier, M. (2016). Environmental changes bridge evolutionary valleys. *Science advances*, e1500921.

Wolpert, D. (1996). The lack of a priori distinctions between learning algorithms. *Neural computation - VIII*, 1341--1390.

Wolpert, D., & Macready, W. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation - I*, 67--82.

Wolpert, D., Macready, W., & others. (1995). *No free lunch theorems for search.*