Modular HyperNEAT

An evolution algorithm that combines evolving modular neural networks using spatial patterns

1st Mengjie Zhang
Department of Electrical and Electronic Engineering
Imperial College London
London, United Kingdom
jerrymym23@gmail.com

2nd Marcus Ghosh
Department of Electrical and Electronic Engineering
/I-X Centre for AI in Science
Imperial College London
London, United Kingdom
m.ghosh@imperial.ac.uk

***Abstract*—The presence of modular structures in biological neural networks has been shown to offer a significant advantage for intelligent animals to perform a multitude of complex functions. The adaptation of this idea in artificial neural networks has also shown to improve traditional networks in terms of the neural network's construction and its subsequent performance in tasks. This paper presents an evolution algorithm that constructs a complete network system of smaller modulized networks using the spatial information of the network and its environment. We then demonstrated this new method offers substantial improvements in solving tasks that involve navigation and image classification. This will provide more evidence and insight on how modularity can be beneficial in evolving complex machine learning systems.**

I. Introduction

Evolution algorithm (EA) is an optimization technique that is commonly used to create mathematical functions and machine learning models for task solving. This technique differs from traditional optimization techniques in that it does not require the computation of an analytical relationship between a model's parameters and its performance metric. Rather, it relies on a "survival of the fittest" idea in which a population of models are initialized first and the best performing ones are selected for reproduction of new populations. The process iterates until a desirable model is produced.

Neuroevolution (NE) is the process of applying EA on artificial neural networks. Since a neural network is fundamentally a mathematical model, each network can be regarded as an individual function in an evolvable population. The evolution process can thus be carried out as in a typical evolution algorithm. There have been many techniques to evolve neural networks. Some involves fixing the network's topology (connection patterns between neurons) and only evolving its weights. Despite this, most widely used methods often involve evolving both the weights and the connections. Among them, the most popular algorithm is called *Neural Evolution of Argumented Topologies* (NEAT) [4] published in 2002. NEAT improves traditional *TAWENN (Topology and Weight Evolving Neural Networks)* networks by grouping each of them into species. This optimizes the evolution process by preventing potentially better solutions from being removed prematurely due to the initially low fitness of the networks.

HyperNEAT [3] is a substantial improvement to NEAT. It employs an implicit encoding of network genomes. The genome of a network describes the information of its neurons (such as activations) and their connectivity. It completely governs how a network would be constructed. Previous method uses explicit encoding of genomes. This means one directly specifies which neurons are connected to which (e.g. *Neuron 1* $\rightarrow$ *Neuron 2*, $w = 0.5$). HyperNEAT uses a *Compositional Pattern Producing Network (CPPN)* to output the weights between 2 neurons. This means the previous tasks of evolving the best network genome now becomes evolving the best CPPN.

II. Related Work

The main challenge in combining modularity with neuroevolution is the simultaneous evolution of module structures and inter-module connections. Many existing NEAT versions that employ modularity treat each modules as a single neuron. Some methods also ignore the evolution of inter-module connections and focus on evolving the modules. The table below summarizes the common approaches used.

Topology	Modules[1]	Typical algorithm(s)
fixed	fixed	G.D.
fixed	evolved	NEAT , ExNEAT [5]
evolved	fixed	EA [1], ModularNEAT [2], HyperExNEAT
evolved	evolved	EA, HyperExNEAT

TABLE I: A summary of common algorithms for modular neuroevolution.

III. Method

Modular HyperNEAT builds upon the HyperNEAT algorithm by using modular structures to construct a network system. Similar to HyperNEAT, a substrate of coordinates is defined to represent the input and output neuron locations of the environment. The input and output neurons of the modules are also represented using coordinates. The connections between the environment's input/output and the module's input/output are both determined by the CPPNs.

The major challenge of this method is the addition of evolving modules as well as the CPPNs. In HyperNEAT, the

[1]Here it means module structures.

sole evolution of CPPN is enough to complete the construction of the final network. In this method, the CPPN needs to work with the modules to produce the optimal final network. This means a new co-evolution algorithm is needed for this method to function properly. The biggest issue in using a co-evolution method is the high computation cost. Since for each individual in population 1, it needs to be tested for compatibility with every individual in population 2. In our work, we used the *Best Partner Sampling* co-evolution strategy. This method samples the best n individuals in population 2 for each individual in population 1. This significantly reduces computation overhead while maintaining a stable, ascending evolution progress.

Algorithm 1 Modular HyperNEAT algorithm

$p_{module} \leftarrow neat.population$
$p_{CPPN} \leftarrow neat.population$
$gen \leftarrow userdefined$
$bestFound \leftarrow False$
$g \leftarrow 0$
while $g < gen$ or $notbestFound$ **do**
  for g_{module} in $p_{modules}$ **do**
    for g_{CPPN} in $bestCPPNs(p_{CPPN}, n)$ **do**
      $g_{module}.fitness \leftarrow g_{modules}.fitness + evalFitness(g_{module}, g_{CPPN})$
    end for
    $g_{module}.fitness \leftarrow g_{module}.\bar{fitness}$
    if $g_{module}.fitness \geq fitnessThresh$ **then**
      $break$
    end if
  end for
  for g_{CPPN} in p_{CPPN} **do**
    for g_{module} in $bestModules(p_{module}, n)$ **do**
      $g_{CPPN}.fitness \leftarrow g_{CPPN}.fitness + evalFitness(g_{CPPN}, g_{module})$
    end for
    $g_{CPPN}.fitness \leftarrow g_{CPPN}.\bar{fitness}$
    if $g_{CPPN}.fitness \geq fitnessThresh$ **then**
      $break$
    end if
  end for
  $p_{module} \leftarrow evolve(p_{module})$
  $p_{CPPN} \leftarrow evolve(p_{CPPN})$
end while

1) Co-evolution strategy: In general, evolution of modules and CPPNs has adversarial effects on the respective population. This is due to the performance need to be measured as a module-CPPN pair. Therefore, evolving only the module or CPPN population could disrupt the fitness progression of the agent.

One of the solution is to sample the partner population while evolving the other population. The co-evolution strategy is called *Best Partner Sampling* and is the key to the proper function of Modular HyperNEAT.

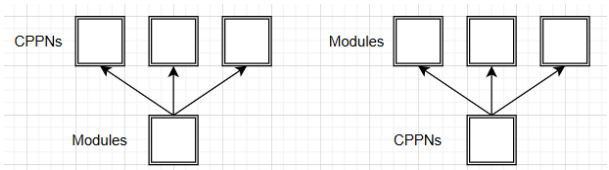

Fig. 1: Illustration of the co-evolution strategy. Each CPPN is evaluate on a group of best performing modules. The same process is applied to the modules.

IV. Results

A. Maze Navigation

For each trial, the agent was tested on 20 mazes to obtain an average performance. The fitness of the agent is calculated as the average time the agent uses to solve the mazes and the mean squared distance between the agent's final location and the goal position.

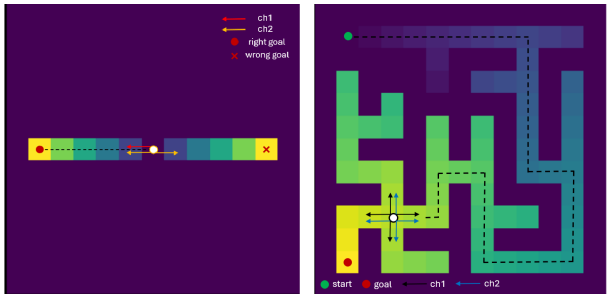

Fig. 2: The maze environment is a 2D plane with pixel coordinate representing paths (a pixel with value 1) and walls (a pixel with value 0). For track (1D) mazes, the agent starts in the middle of the maze and the goal is to reach the correct side. For the 2D maze, the agent starts at one end of the maze and floor of the maze emits sensory cues that leads to a goal position. Each sensory cues are multi-channeled. Meaning the agent will need to utilize multi-modal information.

1) Evolution performance: Training process

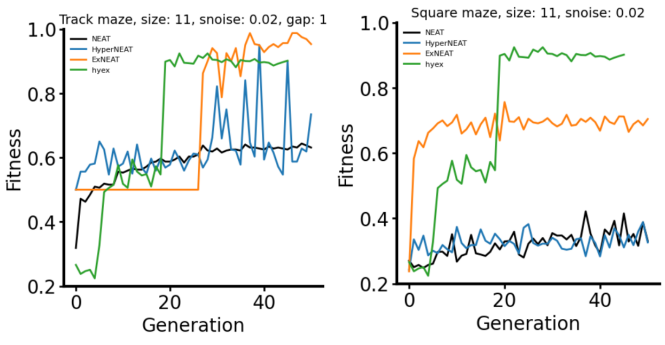

Fig. 3: Training performance comparison of agents in different maze navigation tasks.

Algorithm	1D maze	2D maze	Square	Cross
NEAT	29.87	16.32	23.31	45.66
HyperNEAT	31.74	16.61	24.60	47.12
ExNEAT	35.93	34.60	29	55.02
Modular HyperNEAT	45.06	32.85	28.08	54.38

TABLE II: Performance comparison of different algorithms in terms of evolution score.[2].

2) Discovery of optimal topologies: Maze task

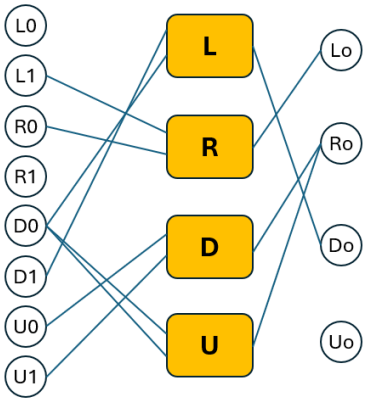

Fig. 4: An example of inter-module topology discovered by Modular HyperNEAT. The algorithm was able to alternative, non-obvious module relations that are optimal in solving the tasks given.

B. Image Classification

There are 2 types of image classification tasks that we used to test the agent. Both have the same goal, to locate the center coordinate of a certain shape in an image. All images were of shape 9×9. The substrate was a state space sandwich with the same connection pattern for the input and output layers. Therefore, there are 6561 possible connections to search through.

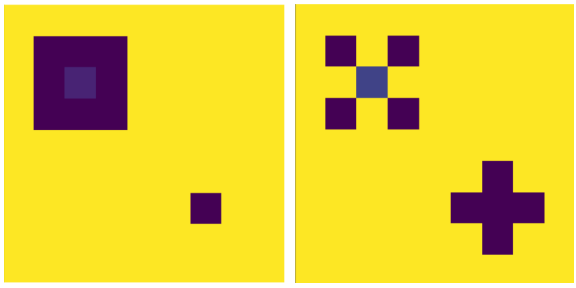

Fig. 5: The first task involves locating the center of the larger square among 2 squares (left) and the second task involves finding the center of the cross between a cross and a plus (right).

[2]The evolution efficiency is defined as the integral of the fitness vs generation graph over generations.

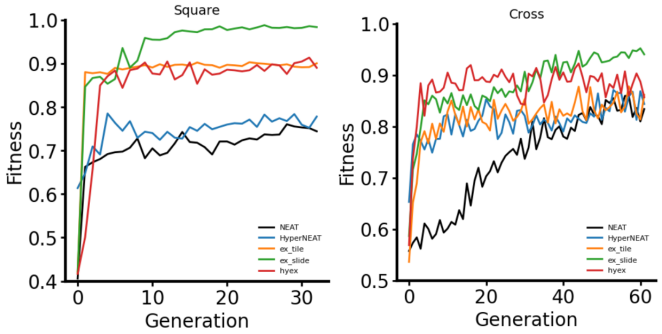

Fig. 6: Training performance comparison of agents in different image classification tasks.

V. Discussion and Conclusion

We have presented Modular HyperNEAT. The novel neuroevolution algorithm managed to out-perform both of its predecessors, NEAT and HyperNEAT. Its flexibility was demonstrated from being able to solve a variety of tasks that involve either time-depending sequential decision making tasks or high dimensional, image classification tasks.

Future improvements could be to co-evolve multiple modules instead of reusing the same module across the network. This would enable the algorithm to form a more complex system, such as a robot. Another drawback of HyperNEAT is inherited from HyperNEAT. That is, the CPPN requires a predefined coordinate pattern to produce optimal performance. While this is totally expected for a pattern producing network, it limits the flexibility for the algorithm to discover underlying patterns on its own. One possible improvement could be to combine ES-HyperNEAT with this algorithm.

VI. References

References

[1] Charlotte Gibson. “Evolving Modular Neural Networks”. In: (2025).

[2] Joseph Reisinger, Kenneth O. Stanley, and Risto Miikkulainen. “Evolving Reusable Neural Modules”. In: *Genetic and Evolutionary Computation – GECCO 2004*. Ed. by Kalyanmoy Deb. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 69–81. ISBN: 978-3-540-24855-2.

[3] Kenneth O Stanley, David B D’Ambrosio, and Jason Gauci. “A hypercube-based encoding for evolving large-scale neural networks”. In: *Artificial life* 15.2 (2009), pp. 185–212.

[4] Kenneth O Stanley and Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[5] Mengjie Zhang. “ExNEAT-Separable network through coevolution of partially-dependent sub-modules”. In: (2025).