# The Utilization of the Hypercube-based Encoding for the Evolution of the Hybrid Artificial Neural Networks

Pavol Sekeres*

Czech Technical University - Faculty of Electrical Engineering
pavol.sekeres@gmail.com

**Abstract**

*In this report I discuss the possibility of adopting the HyperNEAT algorithm for the Hybrid artificial neural networks(HANNs). I present the modification build on the top of the HyperNEAT algorithm. This upgrade allows utilizing the benefits of the known HyperNEAT for the learning of the structure of the HANN. Furthermore I have chosen the concrete HyperNEAT implementation and provided the necessary changes in order for it to operate properly on HANN. I show in the report, how this implementation can be reused for the purpose of learning the Topology of an arbitrary feed-forward HANN.*

## I. Theoretical Background

In this section, I provide the description of the key features of the extended algorithm. The concepts of the Compositional Pattern Producing Networks(CPPN) and NeuroEvolution of Augmenting Topologies(NEAT) will be explained briefly. Furthermore I will show, how the HyperNEAT algorithm utilizes the CPPNs on encoding the structure of a large scale artificial neural network.

### I. Compositional Pattern Producing Network

In biological genetic encoding, the mapping between genotype and phenotype is indirect. The phenotype typically contains orders of magnitude more structural components than the genotype contains genes. For example, a human genome of 30,000 genes (about three billion amino acids) encodes a human brain with 100 trillion connections[1]. The way to develop structures in such large scales would be to implement a mapping, that translates few dimensions into many. A most promising area of research in indirect encoding is developmental encoding, which is motivated from the biology. Development facilitates reusing genes, because the same gene can be activated at any location and any time during the process of development[2]. CPPN's are special type of ANNs which build on top of the idea of the developmental encoding. They are an abstraction of the natural development for a computer running an evolutionary algorithm. Typical CPPN takes on input the coordinates in the Cartesian space and returns a value. This way we can indirectly extract the properties of a large scale system using the order of magnitude smaller CPPN. Furthermore the CPPN is capable of exploiting similar properties, as found in the biological systems. By using the set of different activation functions, it achieves: symmetry, imperfect symmetry and various repetitions with variations in the encoded large scale system.

*A thank you or further information

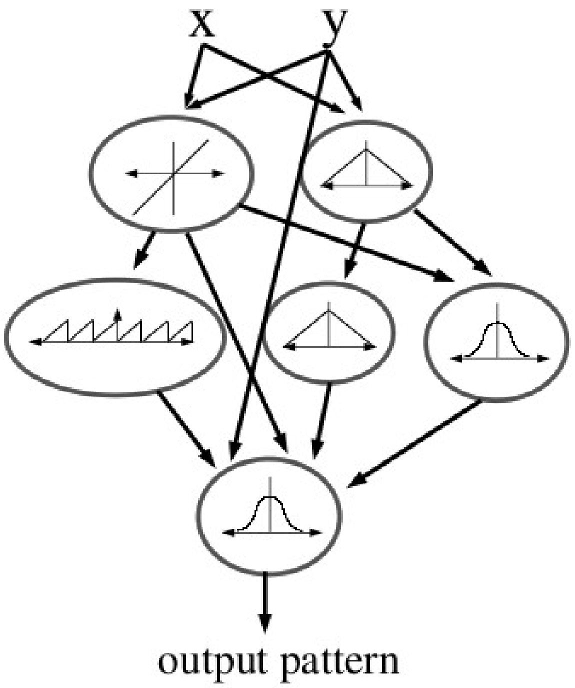

**Figure 1:** *Example of the compositional pattern producing network. In this case, it takes just two coordinates of the Cartesian space[2].*

## II. NeuroEvolution of Augmenting Topologies

NEAT is a neuro-evolutionary method. It begins evolution with a population of small, simple neural networks and complexifies the network topology into diverse species over generations, leading to increasingly sophisticated behavior[3]. By using the historical marking concept, it allows the increased complexity of the networks in the later generations. It builds on the idea, that the structures of the network, which provide similar functionality are being evolved in the same generation. This allows NEAT to perform crossover across different topologies. Another important feature of the NEAT is speciating of the population. The subsets of the population compete primarily in their own niches, instead of with the complete population. This allows the seemingly useless structures to optimize their performance in the network. Historical markings are used to speciate the population. NEAT begins with a uniform population of the simple networks without any hidden nodes, differing only in their initial random weights[2]. It can be easily adopted to the evolution of the CPPNs, by choosing the activation function of the node from the given set of functions.

## III. Hypercube-based NeuroEvolution of Augmenting Topologies

HyperNEAT solves the problem of the mapping of the spatial patterns produced by the CPPN to the connectivity patterns in the neural network. Instead of querying the CPPN for the value of the concrete point in Cartesian space, we can query the CPPN by giving the spatial coordinates of two points. One of them represents the input and the other the output point of the concrete neural connection. In this way, we extract the values of all the weights of the network. Because the connection weights are function of the positions of their source and target nodes, the distribution of the weights on connections throughout the grid will exhibit a pattern that is a function of the geometry of the coordinate system[2]. The connectivity pattern produced by the CPPN is called the substrate. Depending on the task we want to solve, different types of substrate are more or less suitable.

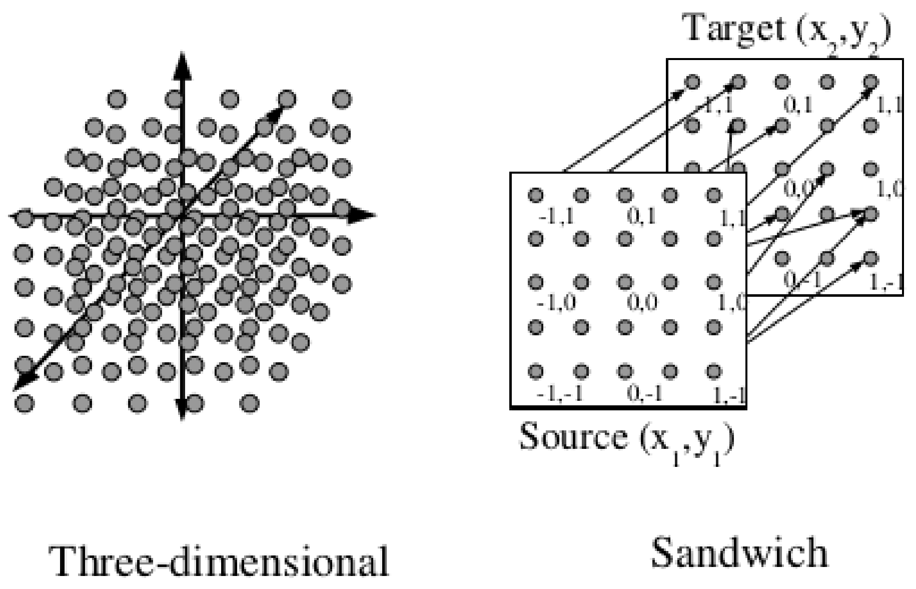

**Figure 2:** *Substrates for HyperNEAT. In general, there are other possible substrates for the HyperNEAT algorithm. However for learning feed-forward HANNs we will utilize only three-dimensional and sandwich substrates [2].*

## II. Proposed Extension of the Algorithm

In this section I provide the exact extensions of the algorithm, which are necessary in order to learn the desired weights of the hybrid artificial neural network. The solution for the different types of substrates will be discussed in detail.

### I. HyperNEAT for HANN

The HANN unlike standard ANN is built up from the smart neurons. These are modules, which receive arbitrary number of inputs and using some implemented algorithm further provide arbitrary number of outputs to the next layer of the network. The algorithms they incorporate can vary greatly. If we want to apply the HyperNEAT principal on HANNs, we must understand, how exactly they differ from the ANNs.

HANNs can be evolved using the evolutionary algorithm with direct encoding[4]. However in case of the large scale HANNs it would be not feasible to use the direct encoding technique. In addition if the pattern encoded by the network exhibits geometrical properties, which the

HyperNEAT provides(symmetry, imperfect symetry, etc.), it could be suitable to use this method instead of an evolutionary algorithm with the direct encoding. In general, the HANNs can contain any type of the neurons in each layer. However in many practical experiments, we do not need more than one type of smart neuron in each layer of the network. For the simplicity, let us focus on the networks, which contain only single type of neurons in the same layer. In the following section we will formulate the algorithm for the different substrates.

## II. Sandwich Substrate

In case of the sandwich substrate, we do not have any hidden layer in the network. This allows us to use the HyperNEAT algorithm without major modifications. For each output of the neuron in the input layer, we create interface neurons, which can contain only single output. For the output layer we again introduce interface neurons with single input. One interface neuron for each input of the output layer neuron. We then query the weights between the interface neurons only. This is done in the same way as in case of the ANNs for the sandwich substrate.

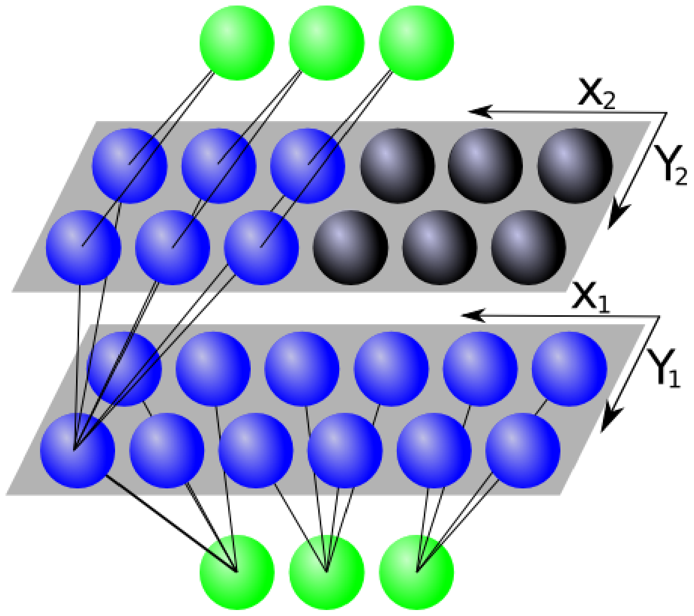

**Figure 3:** *Sandwich substrate of HANN. Green - input and output neurons of the HANN. Blue - interface neurons, they are queried for weights using concrete CPPN, Black - unused neurons, we do not query them, because they do not have matching neurons in the HANN*

## III. Three-dimensional Hypercube(Feed-forward) Substrate

In the three-dimensional hypercube substrate, we query for the weight between $[x_1, y_1, z_1]$ and $[x_2, y_2, z_2]$ coordinates. In the feed-forward case, we allow only connections from the neurons with $z$ to $z + 1$. In this way we get structure similar to the sandwich substrate, but with arbitrary number of layers. Therefore we have to introduce more significant changes to the HyperNEAT algorithm. We will again use the interface neurons, but this time we introduce input and output interface neurons for each smart neuron in any of the hidden layers. The input and output layer of the network will be handled similar to the sandwich substrate case. Additionally, we can reuse the idea about the placement of the interface neurons according to the geometric principal of the CPPNs.

## III. Chosen HyperNEAT Implementation

a In this section I justify the choice of the implementation of the HyperNEAT algorithm, which I will further extend for the purpose of conducting experiments on the hybrid artificial neural networks. From the various implementations of HyperNEAT , which can be found on the HyperNEAT users web page[5], I chose Java library called Another HyperNEAT Implementation (AHNI) [6]. AHNI provides efficient implementation of the algorithm with the possibility of the parallel processing of the evaluation function for the concrete ANN. Because of its object oriented architecture, it can be extended for conducting experiments on custom artificial neural networks. With some additional coding it can serve as the basis for the experiments on hybrid artificial neural networks as well. The parameters of the evolution and evolutionary operators can be modified and different results can be observed.

### I. Proposed Library Extension

AHNI performs several evolution runs until it achieves desired precision of the solution on the given training set, or until the given number of iteration expires. Each iteration starts with the given population or genotype. In this case the genotype stands for the set of the compositional pattern producing networks. These are then transcribed to the phenotype corresponding to the concrete set of substrates with assigned weights. These substrates are then evaluated using the fitness function either sequentially, or in the parallel manor. After that the evolutionary operators are performed on the evaluated individuals and they are pushed to the next iteration of the algorithm. The main loop of the implementation is depicted on the figure bellow.

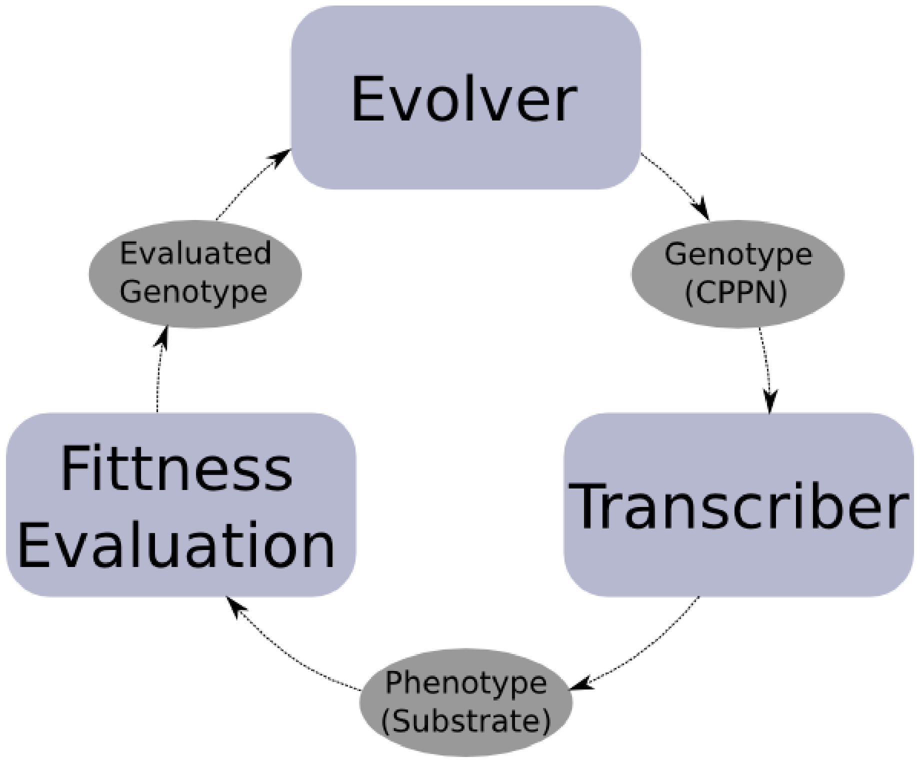

**Figure 4:** *Evolutionary cycle of the algorithm.*

In order to evaluate the hybrid artificial neural networks using given framework, we must first create custom substrate, that would contain the information about the weights of the interface neurons. For the sandwich substrate hybrid artificial neural network, we can use the CPPN with 4 inputs $(x_1, y_1, x_2, y_2)$ , while for the general feed-forward hybrid artificial neural network we can use the CPPN with 6 inputs $(x_1, y_1, z_1, x_2, y_2, z_2)$ ,where z stands for the index of the layer. The framework provides the possibility of evolving both of these CPPN, so we just have to design and implement the transcriber from the genotype to our custom substrate. Furthermore we will code custom evaluation function, which will evaluate the performance of each of the substrates on the given problem. The properties file used in this program allows us to modify the parameters of the evolution, CPPNs and substrates. This allows us to conduct various experiments in order to properly determine the performance of the extended HyperNEAT algorithm.

## References

[1] Deloukas, P., Schuler, G. D., Gyapay, G., Beasley, E. M., Soderlund, C., Rodriguez-Tome, P., Hui, L., Matise, T. C., McKusick, K. B., Beckmann, J. S., Bentolila, S., Bihoreau, M., Birren, B. B., Browne, J., Butler, A., Castle, A. B., Chiannilkulchai, N., Clee, C., Day, P. J., Dehejia, A., Dibling, T., Drouot, N., Duprat, S., Fizames, C., and Bentley, D. R. (1998). A physical map of 30,000 human genes. Science, 282(5389):744 - 746.

[2] Kenneth O. Stanley, David DAmbrosio, Jason Gauci, A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks,Artificial Life journal 15(2), Cambridge, MA: MIT Press, 2009.

[3] Stanley, K. O., and Miikkulainen, Evolving neural networks through augmenting topologies. Evolutionary Computation, (2002)10:99 - 127.

[4] Pavel Nahodil, Jaroslav Vitku, Evolucni architektura chovani umelych bytosti - agentu v danem prostredi, 2014

[5] Kenneth O. Stanley Associate Professor University of Central Florida Dept. of EECS, Computer Science Division webpage: http://eplex.cs.ucf.edu/hyperNEATpage/

[6] Oliver Coleman, Java HyperNEAT implementation https://github.com/OliverColeman/ahni

## Abbreviations

| | |
|---|---|
| **AHNI** | Another HyperNEAT Implementation |
| **ANN** | Artificial Neural Network |
| **CPPN** | Compositional Pattern Producing Network |
| **HANN** | Hybrid Artificial Neural Network |
| **HyperNEAT** | Hypercube-based NEAT |
| **NEAT** | Neural Evolution of Augmenting Topologies |