# An Application of NEAT and HyperNEAT in Solving A Sliding Tile Puzzle

Phuc Hong Ngo
Department of Math and Computer Science, Beloit College, Beloit, U.S.A

**Abstract**. Neuroevolution is a set of algorithms that use evolutionary algorithms to optimize neural networks without much domain knowledge. We analyze Neuroevolution of Augmented Topologies (NEAT) and its extension HyperNeat in this paper. NEAT evolves both the topology and weight values of a network along with novel ideas of applying speciation, tracking genes, and evolving from simple structures. HyperNEAT uses similar techniques to evolve networks but instead of using direct graph encoding as in NEAT, it uses indirect graph encoding. We use a stochastic single-player game, 2048, as the benchmark problem to compare two algorithms' performance. Even though the game is simple, it has the random factor that may pose a challenge in finding a strategy to achieve a high score. The paper analyzes the strategy and the performance of NEAT and HyperNEAT in 2048 with different parameter settings. Furthermore, code and future work are specified at the end of the paper.

**Keywords;** component; formatting; style; styling; insert (keywords)

## 1. Introduction

Game agents are an irreplaceable component in the game industry. They are non-player characters that interact with the player to better the gaming experience. However, most agents nowadays deploy a path-finding algorithm or decision tree [1] to make decisions which leads to a less-human-like and predictable movement [2]. In addition, in some strategic games, an adaptable and progressive agent is desirable since its purpose is to challenge and train the player. For these reasons, achieving a more human AI in-game is a common goal for professional developers.

One way to create a more natural and less repetitive behavior in-game agents is to use the machine learning technique. Many agents for complex games such as Dota2, Minecraft, or StarCraft [3] have been created using machine learning and proven to be able to outperform humans. Some machine learning models that are commonly used to train agents include Convolutional Neural Networks [4], Long Short-Term Memory [4], Reinforcement Learning [5], and Neuroevolution [6].

The video game 2048 was released on 9 March 2014 by Gabriele Cirrulli [7]. The game has a 4x4 grid with some 2 and 4 tiles initially. In every turn, the player can slide all the tiles to move in one of four directions (right, left, up, down). The tiles will keep traveling until they either collide with the edge of the board or another tile. If two tiles with the same value collide, they get merged into one with double the value. A new tile will also spawn randomly in an empty spot with a value of 2 or 4 after the player made a move. The game will end if (1) there is no pair of tiles that could be combined in any direction and (2) the board is  full. A common goal of the game is to reach the 2048 tile. However, players can choose to continue the game and achieve a higher score.

This research aims to develop a game agent that could play 2048 and achieve a score as high as possible using neuroevolution techniques. We will use NEAT and its extension HyperNEAT in this paper.

This paper is organized as follows: Section 2 discusses the relevant literature of methods of training 2048 agents. Section 3 provides a brief introduction to NEAT and hyperNEAT. Section 4 contains the results acquired from experiments, and Section 5 discusses the conclusion.

## 2. Related Work

Two common ways to perceive 2048 are that it is a search problem, or it is a learning problem [8]. If viewed as a search problem, expectiminimax, and minimax have been applied and proven to be effective in 2048.  Otherwise, reinforcement learning techniques like Q-learning, temporal difference learning are employed.

Yun Nie [9] used tree search strategies, minimax and expectimax,  with the addition of pruning to reduce the dimensions' depth. In the paper, they focus on improving the human heuristic. Their work achieved the highest score of 80728 with pruned expectimax which outperformed initial unoptimized algorithms.

Tejaswini Mandava [8] developed a framework and module to interact with 2048 GUI's component. They deploy Q-learning with improved heuristic functions and are able to solve 2048 in 973 moves.

Marcin Szubert [10] prepared n-tuple networks with a temporal difference algorithm to play 2048. The system produces agents with a nearly 98% winning rate on average. Their 1-ply performance is comparable to one of the best computational search-based agents.

Tuponja Boris [11] attempts to use the Neuroevolution of Augmenting Topologies to evolve agents to play 2048. However, the game proves to be a challenge for the method because of the randomness.

Based on the literature review described above, we could have the following research questions: How does HyperNEAT perform in 2048? Is it better than NEAT?

## 3. Methodologies

We set up experiments by using two Python libraries for NEAT and HyperNEAT. Specifically, NEAT-Python [12] for NEAT experiment and PUREPLES [13] (which is a fork of NEAT-Python) for HyperNEAT. We also use Numpy for vectorized operators and tkinter for the game UI. The game UI is my modified fork from qw [14]. The hardware used to conduct the research is Intel i7-8750H CPU 2.20GHz × 12 with 16 GB of RAM.

### *A. NEAT*

NEAT is a neuroevolution algorithm that was developed by Ken Stanley [15]. The algorithm uses the direct encoding of neural networks and the genotype includes two genes: node genes and connection genes. The node genes signify whether a node is an input, a hidden, or an output node. Connectivity genes is a list of edges and their attributes including information about an edge's in and out nodes, weight, innovation number, and if they are enabled or not.

Unlike other neural networks that use fixed topology, NEAT can evolve both its weight and topology. This gets rid of the lengthy process of trial and error by humans to choose the suitable structure for each problem [15]. In addition, the topology chosen by humans won't always guarantee the best results. Along with evolving topology, NEAT also deploys three novel ideas: tracking genes history markers, specification, and complexifying.

The competing convention is defined to be where there is more than one way to present the phenotype or neural network in our cases. The offspring of such parents are likely to be damaged. NEAT overcomes this problem by assigning an innovation number whenever a new gene is added to the population. The innovation starts with 0 and keeps increasing to be assigned to the connections gene. We then can use innovation numbers to align genes during crossover.

Since NEAT allows mutation on its connection, new edges could be added to create new structures. However, such changes are likely to introduce nonlinearity and will get a low fitness level. If they are not protected, they will be eliminated from the population before they could make any innovation or improvement. NEAT uses the technique called speciation which would divide the population into species based on the number of excess genes, disjoint genes, and the average weight differences of matching genes. The genes in one speciation share the same fitness of their speciation. This will allow genes to compete in their speciation to preserve innovation.

Lastly, NEAT reduces its search space and minimalizes the final structure by initializing the population with zero-hidden-node networks. New structures can still be introduced by mutation as discussed. This feature gives NEAT performance advantages compared to other algorithms.

### *B. HyperNEAT*

HyperNEAT uses the same techniques as NEAT. However, instead of using direct encoding, it uses indirect encoding by evolving Compositional Pattern Producing Neural Network (CPPN).

CPPN was proposed by Ken Stanley to evolve large-scale neural networks that make use of the geometric regularities and modularity of the problem [16]. CPNN takes the input of a coordinate in the Cartesian coordinate and outputs the intensity of that point in such space. The difference between CPPN and ANN is that normally ANN just has one fixed function for each layer while CPPN can have different types of functions in different nodes.

But, since CPPN only outputs one value which is spatial information and we need connectivity information, we need to modify CPPN to take two points instead. This CPPN is called a connective CPPN and its output is the weight of the edge between two nodes [17].

HyperNEAT first initialized a population of CPPN with random weight. NEAT techniques are then used to evolve those CPPN. Each CPPN will produce an ANN connectivity pattern which can be used to determine the fitness.

We choose HyperNEAT in this study because 2048 has a geometric symmetry. It can use the information and location of each tile as well as make use of the grid structure of the game.

## 4. Evaluation

To evaluate both algorithms, we must choose the configurations for each algorithms' parameters to achieve the best result. Through experiment, the following settings have been chosen for NEAT and HyperNEAT.

TableI. NEAT Hyperparameters

| Hyperparameters | Value |
|---|---|
| Population Size | 200 |
| Default activation function | sigmoid |
| Activation mutate rate | 0.0 |
| Activation option | sigmoid |
| Connection mutation probability (add/delete) | 0.5 |
| Node mutation probability (add/delete) | 0.2 |

| | |
|---|---|
| Number of nodes (input, hidden, output) | 16, 0 , 4 |
| Compatibility threshold | 1.0 |
| Elitism | 10 |
| Species Elitism | 2 |
| Maximum number of unimproved generations before stagnation | 8 |

TableII. HyperNEAT Hyperparameters

| Hyperparameters | Value |
|---|---|
| Population Size | 200 |
| Default activation function | tanh |
| Activation mutate rate | 0.0 |
| Activation option | gauss, sin, tanh |
| Connection mutation probability (add/delete) | 0.5 |
| Node mutation probability (add/delete) | 0.2 |
| Number of nodes (input, hidden, output) | 32, 0 , 1 |
| Compatibility threshold | 1.0 |
| Elitism | 10 |
| Species Elitism | 5 |
| Maximum number of unimproved generations before stagnation | 8 |

Based on the hyperparameter settings mentioned in Tables 1 and 2 above, we have the following best fitness values and the number of species per generation of each algorithm.

### *A. 20 Generations*

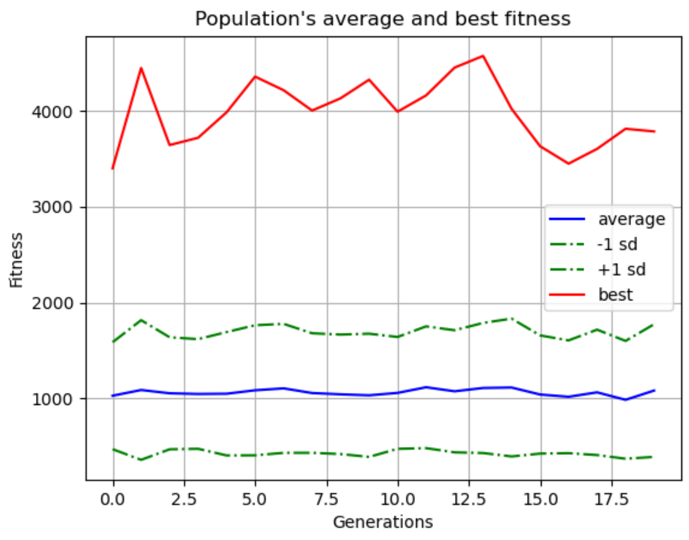

Fig 1. The population's average and best fitness over 20 generations of NEAT

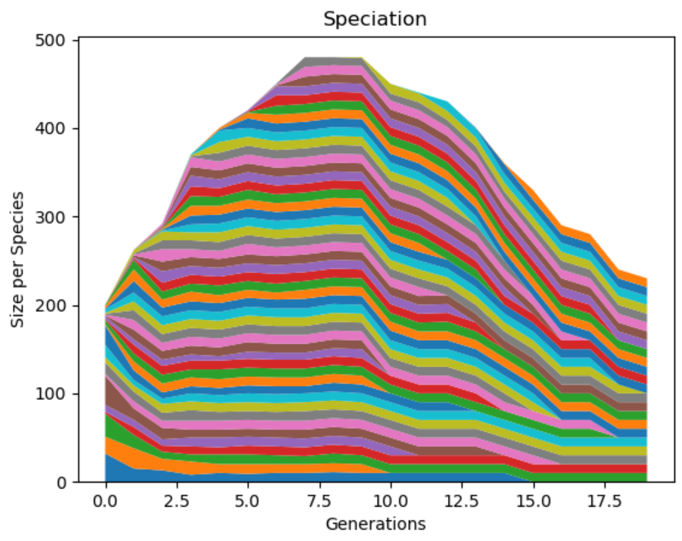

Fig 2. The number of species per generation over 20 generations of NEAT

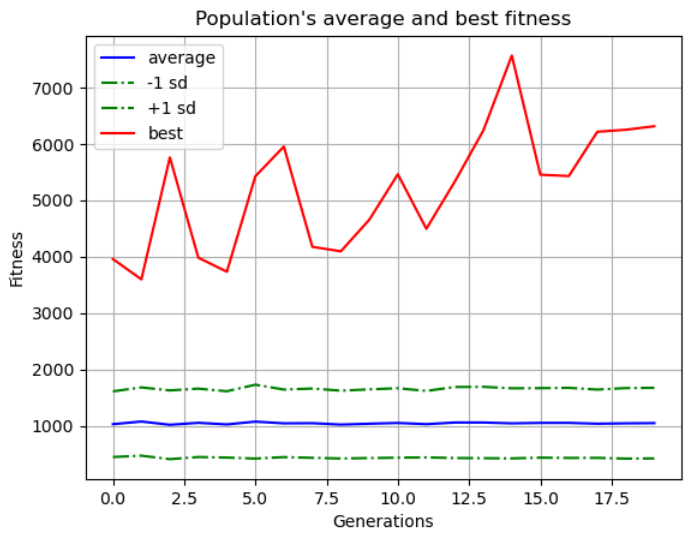

Fig 3. The population’s average and best fitness over 20 generations of HyperNEAT

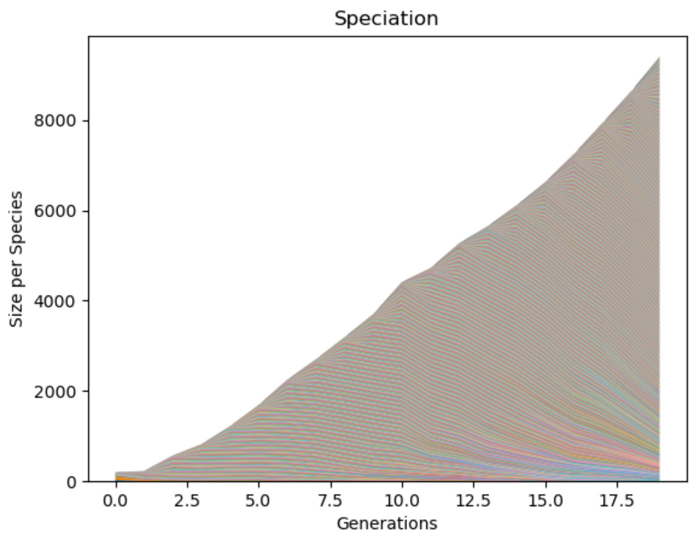

Fig 4. The number of species per generation over 20 generations of HyperNEAT

## *B. 100 Generations*

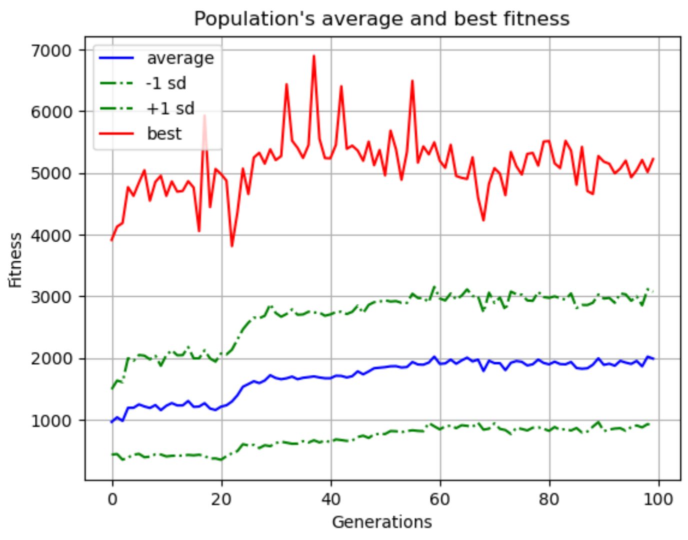

Fig 5. The population's average and best fitness over 100 generations of NEAT

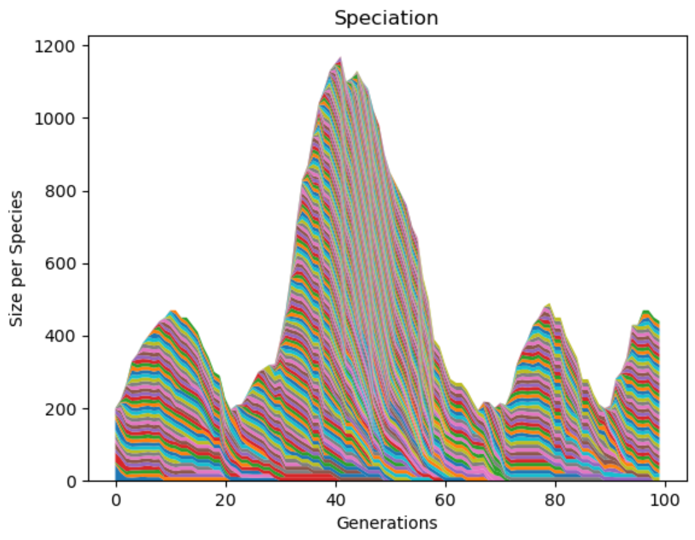

Fig 6. The number of species per generation over 100 generations of NEAT

Firstly, according to Fig.5, we can see that the NEAT algorithm improves dramatically in the first 5-7 generations. However, the best fitness just fluctuates for the future generations even though the average fitness per generation improves. One way to explain this is because of the stochastic nature of 2048. There are many random components like the initial starting board, the location of new tiles, etc

Looking at Fig. 3 and Fig. 1, we can see that HyperNEAT achieves a higher score than NEAT but the average fitness barely improves at all. This could be due to the fact that HyperNEAT often fails to solve tasks that requires complex solutions [18]

## 5. Conclusion

To build an agent that is adaptive and human-like is a common task for both developers and academics. In this paper, we evaluate two neuroevolution algorithms, NEAT and HyperNEAT, on the game 2048. Even though HyperNEAT outperforms NEAT in the first few generations, NEAT generally performs better than HyperNEAT in the long run and the average fitness value does improve which could be due to the stochastic nature of the game. However, due to several limitations, future research with better hardware is needed to be able to run both algorithms with a larger number of generations.

## References

[1] Good, Owen S. (2017). "Skyrim mod makes NPC interactions less scripted, more Sims-like". Polygon

[2] Lara-Cabrera, R., Nogueira-Collazo, M., Cotta, C., & Fernández-Leiva, A. J. (2015). Game artificial intelligence: challenges for the scientific community.

[3] Justesen, N., Bontrager, P., Togelius, J., & Risi, S. (2019). Deep learning for video game playing. IEEE Transactions on Games, 12(1), 1-20.

[4] Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., ... & Silver, D. (2019). Alphastar: Mastering the real-time strategy game starcraft ii. DeepMind blog, 2.

[5] Russell, S., & Norvig, P. (2015). Artificial intelligence: a modern approach (Third edition).

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

[7] https://play2048.co/

[8] Mandava, T., Kakumanu, V., Yarlagadda, Y., & Murthy, P. S. An Efficient Approach to Implement 2048 Game Using Artificial Intelligence.

[9] Rodgers, P., & Levine, J. (2014, August). An investigation into 2048 AI strategies. In 2014 IEEE Conference on Computational Intelligence and Games (pp. 1-2). IEEE.

[10] Temporal Difference Learning of N-Tuple Networks for the Game 2048

[11] Szubert, M., & Jaśkowski, W. (2014, August). Temporal difference learning of n-tuple networks for the game 2048. In 2014 IEEE Conference on Computational Intelligence and Games (pp. 1-8). IEEE.

[12] https://pypi.org/project/neat-python/

[13] https://github.com/ukuleleplayer/pureples

[14] https://github.com/qw/2048-neat

[15] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2), 99-127.

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

[17] Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. Artificial life, 15(2), 185-212.

[18] van den Berg, T. G., & Whiteson, S. (2013, July). Critical factors in the performance of HyperNEAT. In Proceedings of the 15th annual conference on Genetic and evolutionary computation (pp. 759-766).