# NeuroEvolutionary Frameworks to Evolve Chess Evaluation Function

***Abstract*—The IBM Deep Blue's win against Kasparov, the 1997 former Chess World Champion, symbolically proved that Artificial Intelligence (AI) development in autonomous game playing reached, and overcome, human expert skills. With the past years improvements in this research field and the surprisingly growing interest in Chess, this project aligns itself with this scientific landscape providing a novel attempt that leverage a Neuroevolution solution with the purpose of autonomously evolving and tuning populations of virtual players.**

***Index Terms*—Neuroevolution, Chess, CoSyNE, NEAT, evaluation function**

## I. Introduction

For decades, chess has been widely used as a benchmark for leveraging advances in the field of Artificial Intelligence (AI) [6] but only the latest 25 years solutions proved to progressively overcome human skills by outperforming Grandmasters[1] ranking. Chess autonomous playing has been historically considered a frontier for devoloping AI advancements given that it poses a relatively simple task to be model and a finite but rather unfeasibly large search space framework to be perfectly optimised [18].

While a plethora of different strategies had been developed to address the task of autonomously playing chess the most successful implementations were traditionally realized upon engines employing two main components: *a) search algorithm* which recursively explores the tree of possible moves given a current board and an *b) evaluation function* that allows to prune brances of the search space by heuristically estimating the goodness of the current state, thus preventing further explorations of pre-evaluated sub-optimal future states.

Although research on the former component stagnates due to the evolution gained in computational power, advancements on the latter are proving that room for improvement is still possible. Interestingly, evaluations functions, that efficiently predicted whether changes in a board states would provide an advantage or not at the point of reaching Grandmaster levels, were built upon features that were mostly manually tuned. This implies that a collection of a priori information that represents the expected behaviour of each piece in the board, such as King safety or control of the most central positions, had been gathered from chess experts and directly implemented as the initial information to be evolved by implementing evolutionary algorithms yielding promising results [3], [20].

Chess evaluation functions with little to none prior instilled knowledge had been already addressed by evolving the weights of three Artificial Neural Networks (ANN) scanning different board portions by Fogel et al. [7]. Even though it may not perfectly resemble a pure knowledge-free approach, it had been proved that randomly initialised evaluation functions based over ANNs were able to evolve by mimicking the results from the best move gathered by an external chess engine [4].

Despite an initial interest in the early 2010s to solve this task by Evolutionary Programming (EP) methods, recently, attempts like supervised-learning deep residual networks [11], knowledge free self-learning autoencoder-based architectures [2] and deep reinforcement learning solutions [15] may suggest that future trends in this research landscape would further explore Deep Learning applications rather than evolutionary methodologies.

The present study aims at developing a knowledge-free evaluation algorithm that solely evaluates board states by means of *Neuroevolution*, an outstanding evolutionary technique that allows ANN to evolve and search over the space of possible combinations of connections to accomplish various optimization tasks. The present implementation is available at [9].

## II. Methods

### A. Chess Engine

In order to test the present objective a search engine [5] has been gathered and adapted to support the evolution of the populations of evaluation functions. Following a brief description of the features implemented:

- *Iterative deepening* which performs a variable depth search *alpha-beta* pruning over the combinations of choices by discarding branches that could not provide a better outcome according to the current player optimisation objective [12].
- *Move ordering* to adapt the iteraive deepening search to explore branches that are expected to provide an higher evaluation for first.
- *Transposition table* which dynamically saves past moves by exploiting Zobrist hash tables to speed up the seach over new brances [21].
- *Quiescence Search* and *Null move* which provides an expansion over the currently set search horizon in order to avoid disruptive outcomes and to reach a safe position [1].

For the present study the chess engine was set to look up to 4 pyles depth allowing to run with a time limit of 3 seconds in order to have a suitable trade off between the number of moves explored and computational cost.

### B. Board evaluation function

The evaluation function is supposed to provide a representative score according to a given board state in a heuristical

[1]Players awarded by the greatest World Chess Organization FIDE title.

manner. It merges information from the sum of all pieces' *material values* and a *positional estimate* which seeks to describes the state of the board by assigning either a positive or negative score whether the match is currently dominated by the White or the Black player respectively. The material values are the following: $\{m_{Pawn} = 100,\ m_{Knight} = 320,\ m_{Bishop} = 330,\ m_{Rook} = 500,\ m_{Queen} = 900,\ m_{King} = 10000\}$ and are to be considered either positive or negative with respect to the side for the White and Black respectively. The King does not contribute to the total amount of the overall pieces' material values since it is the only one that can not be captured. Given a board state ($b$) the single $i-th$ genome evaluation function is defined as:

$$f_{TOT}(b) = \frac{1}{m_{Queen}} \sum_{i=1}^{N_{pieces}} m_i + f_{position}(b)$$

where the sum of all $m_i$ pieces is rescaled by the highest value of the piece which can be captured (the Queen) and $f_{position}(b)$ defines the positional values function described in details as follows.

### C. Fitness definition

The fitness chosen for the present work relies on a high behavioural consequence gained from the outcome of matches played against opponents: the $i-th$ genome is evaluated according to the score gained in a tournament of $M$ number of matches playing against virtual players from the same generation $g$:

$$fitness_i^{[g]} = \sum_{n=1}^{N_{matches}} score_{(i,n)}$$

in which $score = \{1, 0.5, 0\}$ whether it wins, draws or loses respectively. The rationale of the fitness relies on the assumption that the higher the tournament score of a given virtual player, the more likely its genome is to evaluate the current board state to gain advantage over the opponent while performing the Iterative deepening search to choose best moves to be played and, consequently, to win the match.

$M$ (`number_of_matches`) is an hyperparameter regulating the number of matches a single genome will play in each generations' tournament for each side. The higher this value, the higher the computational cost due to the high number of matches to be played and the more likely each player is to encounter more opponents, thus influencing the magnitude of the final score. In the current study the parameter had been set to allow each player to perform 6 matches in total (3 per each side).

Each tournament's match had been implemented to be executed by a single CPU's thread by the `ray` Python library allowing to spawn multiple matches at the same time.

## III. Neuroevolution

Two attempts were made in order to design neuroevolution frameworks with the purpose of evolving evaluations functions that are respectively:

- Evolving exclusively the weights and biases of fixed ANN architectures by *Cooperative Synapse Neuroevolution* (CoSyNE) [8].
- Evolving the weights and the topology of different ANN architectures species by means of *Neuro Evolution of Augmented Topology* (NEAT) [17].

### A. CoSyNE

With the purpose of mapping each ANN synaptic connection weights as a subpopulation, CoSyNE provides a framework that allows to enforce coevolution by coperatively combining subgenotypes in order to exchange subgenomes of less fit individuals to those of most fit individuals to preserve the chance of exploring a wide range of solutions [8].

*1) **Positional value function**:* The function is estimated by:

$$f_{position}(b) = \tanh \prod_{l=1}^{L} \sigma(W_i^{[l]} \times A_i^{[l-1]} + bias_i^{[l]}) \in (-1, 1)$$

and corresponds to the results gained by the product of the $l = 1, \ldots, L$ ANN layers considering the matrix of weights $W_i^{[l]}$ and $bias_i^{[l]}$ of the $i-th$ genome at layer $l$ and for which the initial input $b$ ($X_i = A_i^{[0]} \in \mathbb{Z}^{64}$ ) is defined by the $8 \times 8$ flattened chess board filled with the material values of each piece in its corresponding place. For each hidden layer a $sigmoid$ ($\sigma(x) = 1/(1 + exp(x))$ activation function had been implemented allowing the output layer returned the board positional evaluation in a $(-1, 1)$ range by the $\tanh$ activation function.

*2) **Evolution parameters**:* To efficiently model the evolutionary procedure, a population of 16 virtual players, whose genomes are represented by a set of real-values vectors storing the "synaptic" information that will be further used in the board evaluation function as weights and biases terms to be used in the ANN, had been initialized. Selection pressure, along with crossover and mutation were applied jointly with parent genome permutation to allow coevolution through generations. An explanation of the evolutionary framework implemented is described as follows.

**Architectures**. Two architectures were tested in order to model the positional value function $f_{position}$ by means of ANN: *a)* a single hidden layer perceptron ($L = 1$) with $n_{nodes}^{[1]} = 128$ hidden nodes as in the Kolmogorov's theorem[2] [19], [20], and a *b)* multiple hidden layers perceptron ($L = 2$) with $\{n_{nodes}^{[1]} = 64, n_{nodes}^{[2]} = 64\}$ and a single output node. Both attempts were built in order to have a similar amount of parameters in each architecture which were namely modelled by matrices of weights (8320 and 8256 values overall respectively) and bias vectors (128 values). The only distinction consists in the different number of hidden layers involved.

**Network initialization**. Weights were randomly sample from an Uniform distribution according to the Xavier normalization $U(\pm\sqrt{6/(n^{[l-1]} + n^{[l]})})$ where $n^{[l-1]}$ and $n^{[l]}$ are the number of nodes of the previous and the current layer respectively for

[2]According to the Kolmogorov's theorem the number of hidden inputs should be at least twice the number of the input in a shallow multi layer perceptron.

better convergence.

**Selection**. At each generation the best $n/2$ players were selected according to the score they reached and considered as the next generation's parents for reproduction.

**Crossover and Mutation**. Offsprings were generated by performing *uniform* crossover allowing to swap the genome components with probability $p_{swap} = 0.30$ and selecting parents to reproduce with $p_{cross} = 0.60$. A gaussian mutation had been applied with $p_{mutation} = 0.60$ allowing $\sigma_0 = 0.01$ to exponentially decay over generations $g$ by $\sigma_g = \sigma_{g-1} \times 0.995^g$.

**Parent permutation**. Each $i$ parents' genome had been adaptively permutate at each generation within the same synaptic sub-population $j$ in order to have the least fit individuals more likely to swap synapses and those combination of synapses that received higher fitness less likely to change their genome as described by Gomez et al. [8] according to the following probability:

$$p(x_{i,j}) = 1 - \sqrt{\frac{fitness(x_{i,j}) - fitness^{min}}{fitness^{max} - fitness^{min}}}$$

for which $fitness^{max}, fitness^{min}$ refers to the max and min fitness achieved by all parents on that generation respectively.

### B. NEAT

It has been unanimously considered a milestone in the framework of Topological and Weight Evolving Artificial Neural Networks (TWEANNs) due to its ability to keep trace of the topological evolution while performing crossover over different ANN architectures in order to identify homologous structures by the so-called *historical markings* [14]. It allows to progressively speciate different evolved substructures from a minimal network representation in niches to preserve the optimization of newly evolved solution from the evaluation against already optimised solutions.

*1) **Positional value function**:* The function is estimated as in the CoSyNE section with the exception that the network topology may change over generations hence allowing to spawn a not-completely full feedforward connection hierarchically organized as the aforementioned ANN. By contrast, the resulting output still ranges between $(-1, 1)$ allowing thus to be consistent with the magnitude of the evaluation function of the previous attempt.

*2) **Evolution parameters**:* 16 virtual players had been initialized by tuning the *genomic distance* [3] ($\delta$ = 4.0) in order to have all genomes being part of the same initial spiece.

**Architecture**. To be consistent with the previous attempt a single hidden layer network architecture had been initialized with $n = 128$ hidden nodes.

**Selection**. In each generation the whole number of individuals from the past generation had been replaced by offsprings without retaining any parents (no elitism involved).

**Network initialization**. Weights and biased had been intialize randomly from $N(0, 1)$.

[3]With *genomic distance* the similarity between genomes is computed by estimating the degree to which homologous nodes and connections have been evolved and diverged from their common ancestor.

**Network mutation**. With $p_{add} = 0.50$ offsprings' networks can add and remove new connection and add or remove nodes with $p_{node} = 0.20$. More information regarding the NEAT hyparameters settings are available in the `config_neat` file.

### C. Evaluation

Given that the evolutionary process has been designed adopting a fitness function that can not provide an absolute measure of each individual performance, both CoSyNE and NEAT frameworks were tested against the latest *Stockfish* release (v.16) [10], an OpenSource Chess Engine which had been adapted in order to have its performance significantly limited to fill the efficiency gap with the chess engine implementation here presented and to compare both opponents exclusively with respect to their evaluation functions. Specifically, Stockfish had been forced to play by setting its ELO ratings at 400, which is usually associated with beginner-like player capabilities, with few tree search resources (measured in search time limit which had been set to 50 ms) and without a pre-computed hash table of evaluated moves.

Each individual virtual player of a given generation was tested against Stockifish in a 6 match tournament playing 3 matches for each side. Outcomes and the overall number of moves taken were saved for evaluation purposes.

## IV. Result

A qualitative assessment of the choice made by CoSyNE players proves that, regardless the ANN architecture, opening moves were generally oriented to directly attack the opponent King without evaluated any further consequence (premature loss of the Queen while playing as White) (*Figure* 1). However, while playing as Black, virtual players managed to build basics chess openings providing set of choices that better resemble real beginner level chess engines.

In order to evaluate whether coevolution successfully enforced changes in the synaptic representation throughout the evolution process a PCA reduction analysis had been applied over the CoSyNE genotypes to reduce the original high dimensional genotype space in 2 dimensions (*Figure* 2, 3). Across generations it can be noticed how the average mutual distance between individual genotypes tends to decrease in both architectures tested so far: the degree to which this happen seems to resemble an inverse exponential function yielding a steep average distances decrease in the initial generations that progressively saturates (*Figure* 4).

Despite its severe downgrade, neither CoSyNE nor NEAT evaluation functions based chess engines were found to be able to consistently defeat Stockfish throughout generations (*see Figure* 5). Interestingly more evolved generations from the single-hidden layer architecture were found to be less able to beat Stockfish (*Table* I). Also for those players that performed the board positional value evaluation by a two hidden layer architecture resulted in rather similar outcome as shown in *Table* II. NEAT based players, despite achieving no wins, gained the highest number of draws in all generations evaluated (*Table* III).

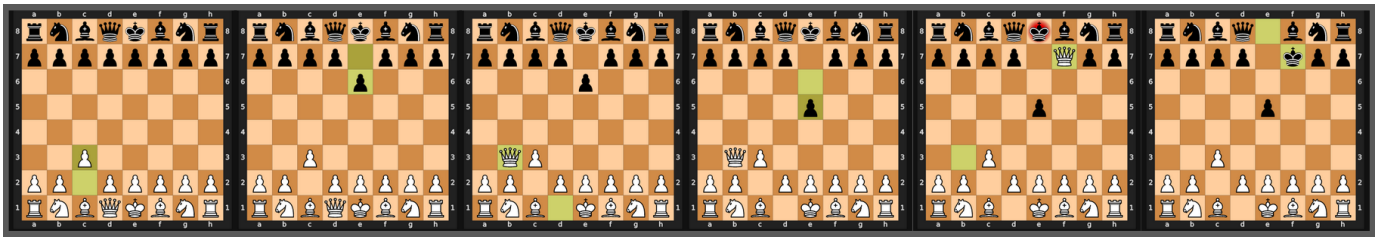

Fig. 1. Example of a greedily proceding opening played by CoSyNE (White) that attempted to directly attack the opponent's King with the fewest number of moves (c3 – e6, Qb3 – e5, Qxf7! - Kxf7).

In order to evaluate whether more evolved generations would have required an increased number of moves before being defeated by Stockfish, all individuals which did not achieve any draw or win were filtered and the averaged number of moves was estimated (*Figure* 6, 7). Each distribution of moves had been tested against with each other in a multiple t-test assessment across generations by considering the amount of moves taken while the virtual players were playing exclusively as White or Black. No significant result was found while looking for differences between number of moves played from players belonging to different generations.

Moreover, virtual players were found to take more moves, and thus yielding statistically significant longer matches, while competing against stockfish playing as Black compared to White: this effect is constant througout all generations and regardless the ANN architecture. This can not be said for the NEAT approach, resulting in only two generations that were found to differ significantly (125 and 200) (*Figure* 9).

## V. Discussion

Performance from virtual players that implemented synapses from the CoSyNE framework was not found to be consistent to those of the reference chess engine: this may allow to state that the proposed evaluation function approach can be ranked according to a $< 400$ ELO rating considering the current set of evolutionary parameters used.

The parent sub-population permutation had been implemented by facilitating synapses exchange over the less fit individual in each generation, preserving most fit genotypes from changes. Although this approach may ensure most successful player to be selected a possible drawback might be that the exploration of different solutions is hindered. This can partially explain the fact that no significant improvement had been found over the most evolved generations.

Throughout the evolution process, NEAT genomes underwent extinction several times due to stagnation and were entirely replaced by new individuals. This may suggest that *a)* the small population size did not facilitate speciations that could have preserved and optimise different architectural niches through generations and *b)* a more appropriate fitness function needs to be defined to better differentiate genomes performances.

The number of moves played by CoSyNE derived evaluation functions while playing as Black were proved to outperform those while playing for the opposite side. Given this counter-intuitive result it can be state that the implementation of the positional value function needs to be further tuned in order to avoid biased negative evaluations which prevents correct search space pruning for the White player only.

The strength of the results here shown could have finally be improved by repeating the evolutionary procedure and averaging the performance outcomes over a given amount of trials.

### A. Difficulties

In principle the chess engine would have allowed to save computation time by means of Hash table implementation: however, given that each individual genotype is supposed to change over generations due to the selection and mutation process, building an extensive hash representation for each genotype would have required an unfeasible amount of time. Additionally, for much more trivial games like Dama a human comparable performance gained through evolutionary process can take more than 5000 generations to be reached [13]. This implies that a much more efficient implementation is needed to achieve such a high number of generations in a feasible amount of time.

Unfortunately, NEAT had been chosen among its most recent variants like *HyperNEAT* or *Evolvable-Substrate HyperNEAT* (ES-HyperNEAT) [14] implementing *Compositional Pattern-Producing Networks* (CPPN) [16] due to the lack of a clear documentation and since it had not been found to be able to correctly estimate any $f_{position}(b)$ evaluation[4]. This could have implicitly boosted the present attempt given the significantly high number of parameters to be evolved.

## VI. Conclusion

To the best of my knowledge, this study accomplish the first attempt to apply NEAT to tune a chess evaluation function. Despite the results here presented, this solution provides room for improvement given that, beside the novelty, more efficient and promising evolutionary frameworks adopting substrate evolution techniques may provide further useful insights in the field of chess autonomous playing.

[4]The current ES-HyperNEAT implementation was found always to yield $f_{position}(b) = 0$ regardless the board state $b$ provided for each individual player in the population. This implies that the final $f_{TOT}$ evaluates according to the material values only.

## REFERENCES


[1] Don F Beal. "A generalised quiescence search algorithm". In: *Artificial Intelligence* 43.1 (1990), pp. 85–98.

[2] Omid E David, Nathan S Netanyahu, and Lior Wolf. "Deepchess: End-to-end deep neural network for automatic learning in chess". In: *Artificial Neural Networks and Machine Learning–ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II 25*. Springer. 2016, pp. 88–96.

[3] Omid E David et al. "Genetic algorithms for evolving computer chess programs". In: *IEEE transactions on evolutionary computation* 18.5 (2013), pp. 779–789.

[4] Omid David-Tabibi, Moshe Koppel, and Nathan S Netanyahu. "Expert-driven genetic algorithms for simulating evaluation functions". In: *Genetic Programming and Evolvable Machines* 12 (2011), pp. 5–22.

[5] Disservin. "github: https://github.com/Disservin/python-chess-engine/tree/master". In: ().

[6] Nathan Ensmenger. "Is chess the drosophila of artificial intelligence? A social history of an algorithm". In: *Social studies of science* 42.1 (2012), pp. 5–30.

[7] David B Fogel et al. "A self-learning evolutionary chess program". In: *Proceedings of the IEEE* 92.12 (2004), pp. 1947–1954.

[8] Faustino Gomez et al. "Accelerated Neural Evolution through Cooperatively Coevolved Synapses." In: *Journal of Machine Learning Research* 9.5 (2008).

[9] gzemo. "https://github.com/gzemo/neuroevolved-chess-engine". In: ().

[10] "https://github.com/official-stockfish/Stockfish". In: ().

[11] Reid McIlroy-Young et al. "Aligning superhuman ai with human behavior: Chess as a model system". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 1677–1687.

[12] Aayush Parashar, Aayush Kumar Jha, and Manoj Kumar. "Analyzing a Chess Engine Based on Alpha–Beta Pruning, Enhanced with Iterative Deepening". In: *Expert Clouds and Applications: Proceedings of ICOECA 2022*. Springer, 2022, pp. 691–700.

[13] Banaz Anwer Qader, Kamal H Jihad, and Mohammed Rashad Baker. "Evolving and training of neural network to play DAMA board game using NEAT algorithm". In: *Informatica* 46 (2022).

[14] Sebastian Risi and Kenneth O Stanley. "An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons". In: *Artificial life* 18.4 (2012), pp. 331–363.

[15] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[16] Kenneth O Stanley. "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic programming and evolvable machines* 8 (2007), pp. 131–162.

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

[18] Stefan Steinerberger. "On the number of positions in chess without promotion". In: *International Journal of Game Theory* 44.3 (2015), pp. 761–767.

[19] Kevin Swingler. *Applying neural networks: a practical guide*. Morgan Kaufmann, 1996.

[20] Eduardo Vázquez-Fernández, Carlos A Coello Coello, and Feliú D Sagols Troncoso. "Assessing the positional values of chess pieces by tuning neural networks' weights with an evolutionary algorithm". In: *World Automation Congress 2012*. IEEE. 2012, pp. 1–6.

[21] Albert L Zobrist. "A new hashing method with application for game playing". In: *ICGA Journal* 13.2 (1990), pp. 69–73.


## APPENDIX A

TABLE I
COSYNE EVALUATION FUNCTION (SINGLE HIDDEN LAYER ARCHITECTURE) OUTCOME IN PERCENTAGE OF MATCHES AGAINST STOCKFISH.

| Generation | Loss | Draw | Win |
|---|---|---|---|
| 25 | 0.9583 | 0.0104 | 0.0312 |
| 50 | 0.9687 | 0.0104 | 0.0208 |
| 75 | 0.9687 | 0.0104 | 0.0208 |
| 100 | 0.9583 | 0.0208 | 0.0208 |
| 125 | 0.9687 | 0.0104 | 0.0208 |
| 150 | 1.0000 | 0.0000 | 0.0000 |
| 175 | 0.9687 | 0.0208 | 0.0104 |
| 200 | 0.9895 | 0.0000 | 0.0104 |

TABLE II
COSYNE EVALUATION FUNCTION (MULTI HIDDEN LAYER ARCHITECTURE) OUTCOME IN PERCENTAGE OF MATCHES AGAINST STOCKFISH.

| Generation | Loss | Draw | Win |
|---|---|---|---|
| 25 | 0.9687 | 0.0312 | 0.0000 |
| 50 | 0.9375 | 0.0520 | 0.0104 |
| 75 | 0.9687 | 0.0208 | 0.0104 |
| 100 | 0.9687 | 0.0312 | 0.0000 |
| 125 | 0.9479 | 0.0520 | 0.0000 |
| 150 | 0.9583 | 0.0416 | 0.0000 |
| 175 | 1.0000 | 0.0000 | 0.0000 |
| 200 | 0.9895 | 0.0104 | 0.0000 |

TABLE III
NEAT EVALUATION FUNCTION OUTCOME IN PERCENTAGE OF MATCHES AGAINST STOCKFISH.

| Generation | Loss | Draw | Win |
|---|---|---|---|
| 25 | 0.8854 | 0.1145 | 0.0 |
| 50 | 0.8750 | 0.1250 | 0.0 |
| 75 | 0.8541 | 0.1458 | 0.0 |
| 100 | 0.8958 | 0.1041 | 0.0 |
| 125 | 0.8958 | 0.1041 | 0.0 |
| 150 | 0.9687 | 0.0312 | 0.0 |
| 175 | 0.8958 | 0.1041 | 0.0 |
| 200 | 0.9479 | 0.0520 | 0.0 |

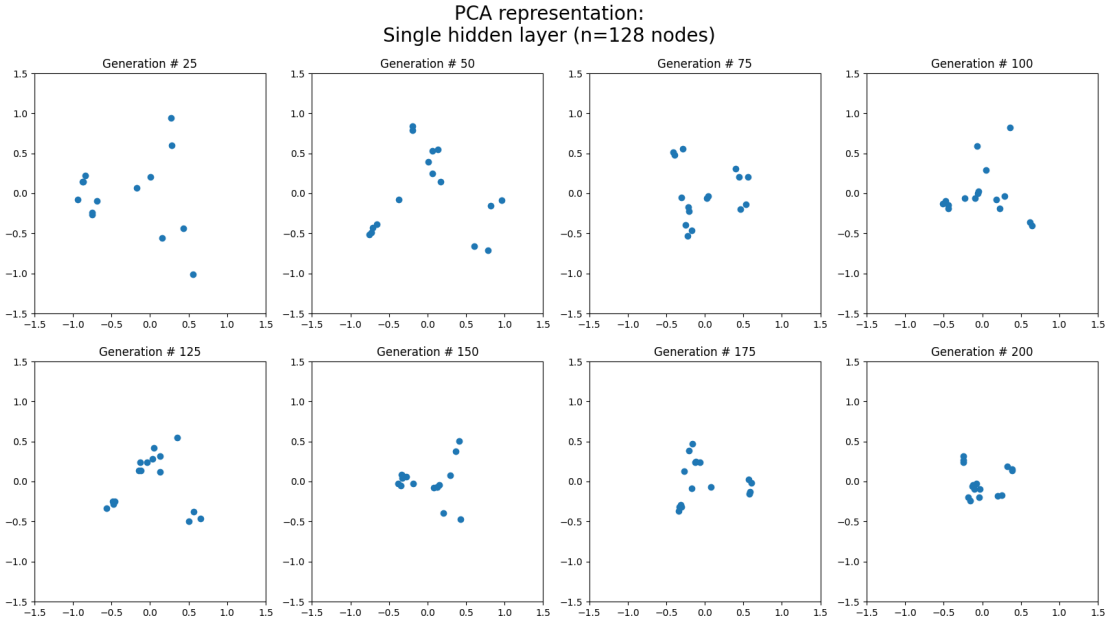


Fig. 2. Two-dimensional PCA representation of the CoSyNE synapses through generations. Each point represent the two dimensional reduced representation of the original 8320 synapses vector representation (considering exclusively the ANN weights from the single layer architecture) of a given virtual player's genome.

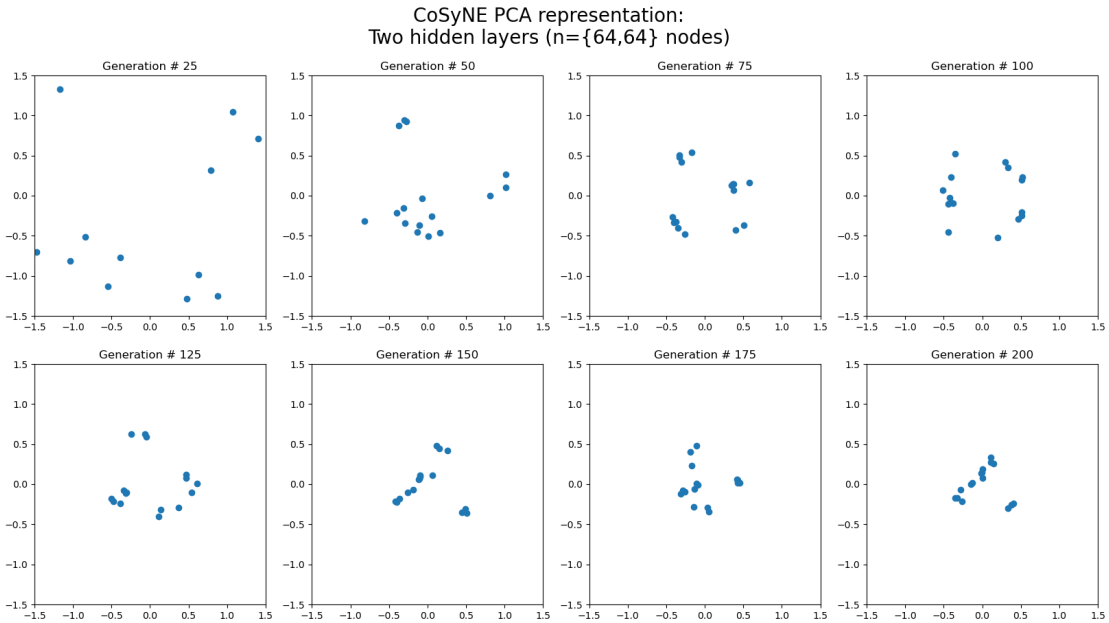


Fig. 3. Two-dimensional PCA representation of the CoSyNE synapses through generations. Each point represent the two dimensional reduced representation of the original 8256 synapses vector representation (considering exclusively the ANN weights from the two hidden layer architecture) of a given virtual player's genome.

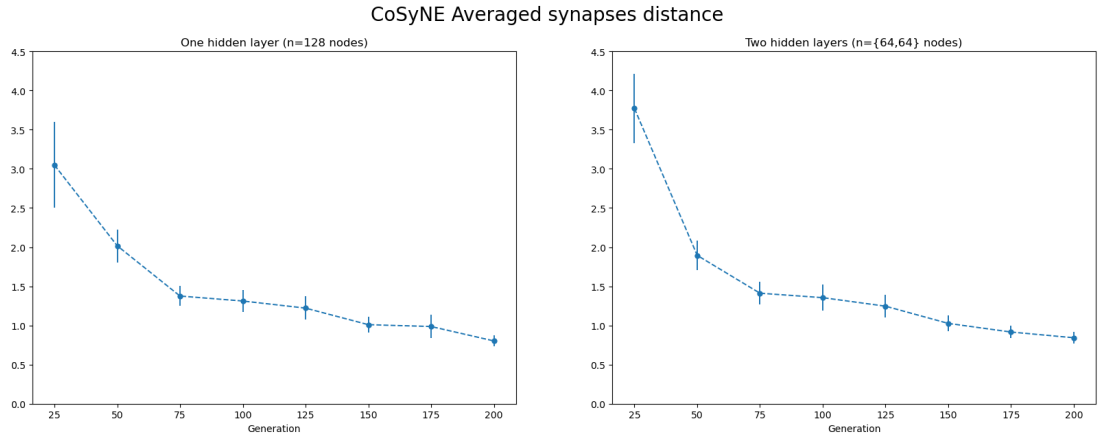


Fig. 4. Mean intra cluster distance representing the averaged synaptic distance across generations.

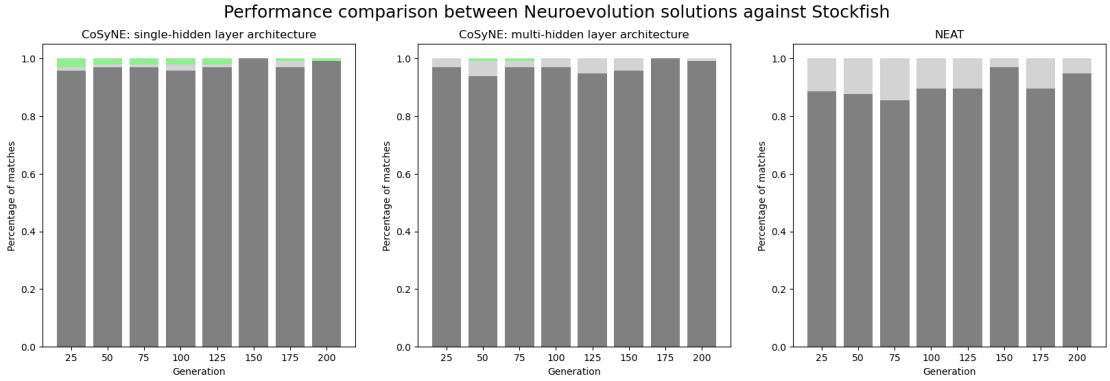


Fig. 5. Comparison of performance achieved by CoSyNE single-hidden layer *Left*, CoSyNE multi hidden-layers *Center* and NEAT *Right* playing $N_{trials} = 6$ matches against Stockfish. Colors specify the percentage of matches outcome (win: green, draw: gray, loss: black).

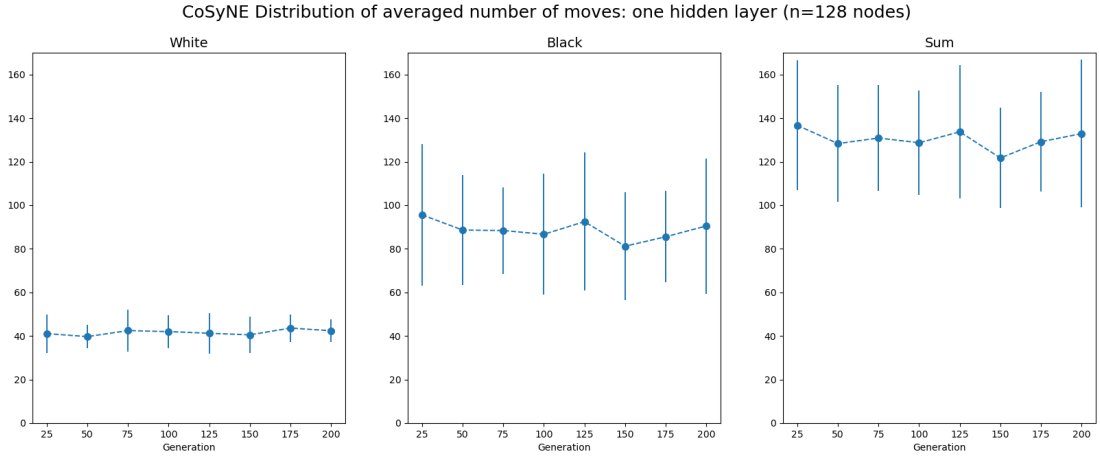


Fig. 6. Distribution of number of moves taken by the CoSyNE single hidden layer ANN architecture evaluation function before being defeated by Stockfish. *Left and Centre:* moves count given that the virtual player had the White and Black side respectively; *Right:* sum of all moves.

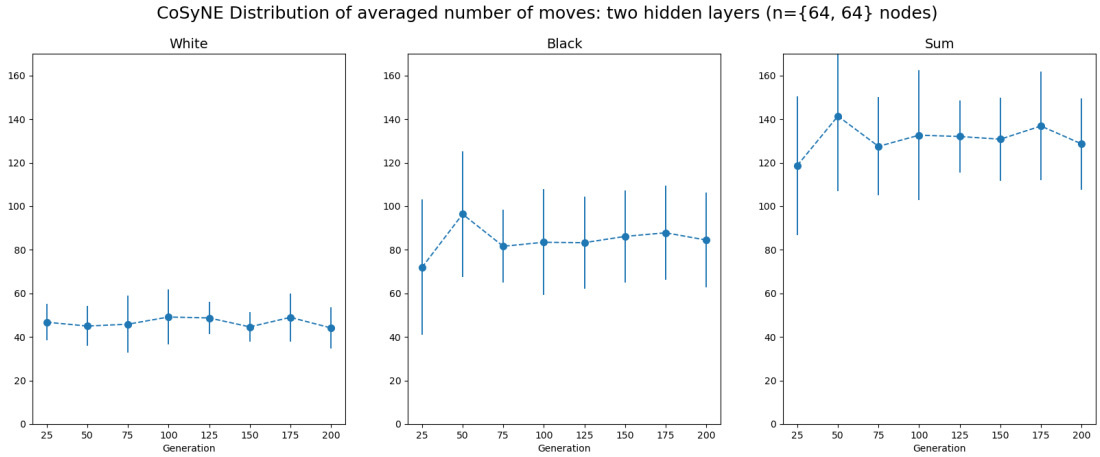


Fig. 7. Distribution of number of moves taken by the CoSyNE multi hidden layers ANN architecture evaluation function before being defeated by Stockfish. *Left and Centre:* moves count given that the virtual player had the White and Black side respectively; *Right:* sum of all moves.

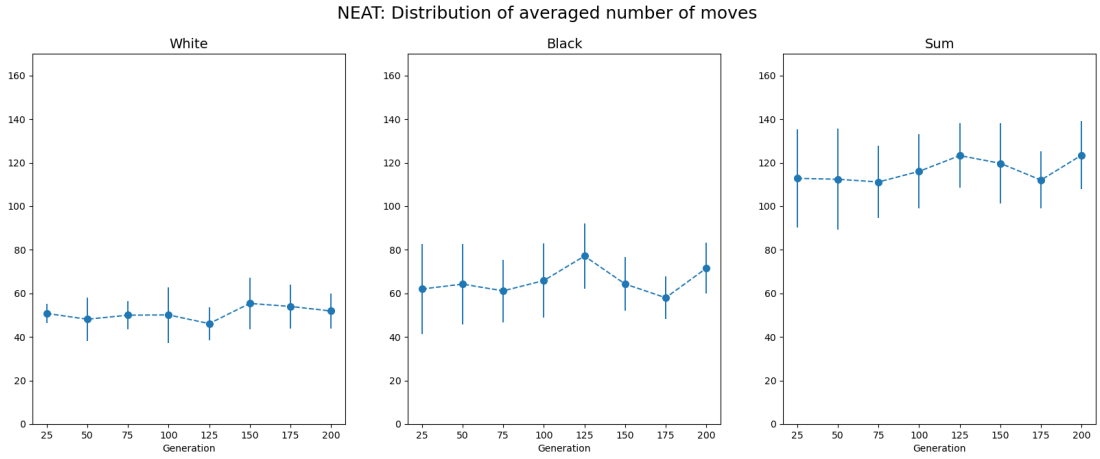


Fig. 8. Distribution of number of moves taken by the NEAT evaluation functions before being defeated by Stockfish. *Left and Centre:* moves count given that the virtual player had the White and Black side respectively; *Right:* sum of all moves.

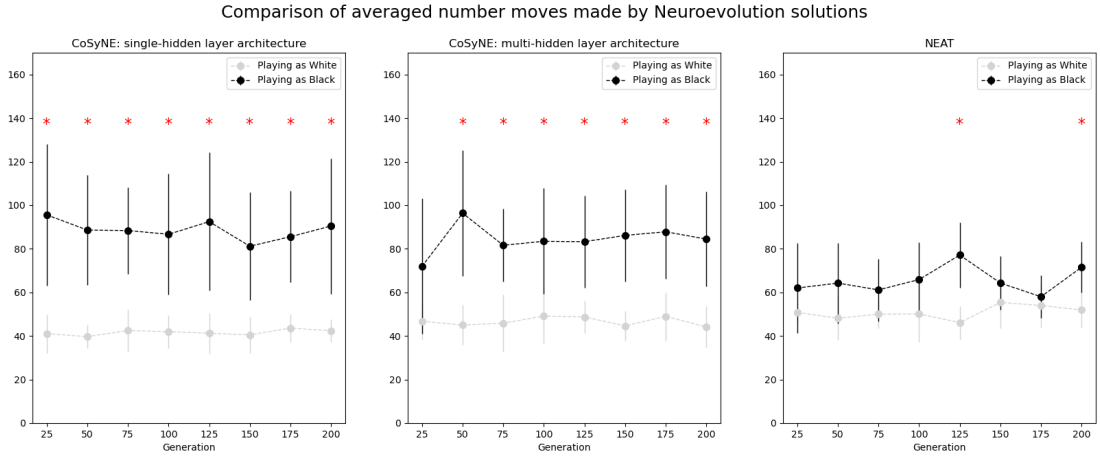


Fig. 9. Comparison of the averaged number of moves played against Stockfish by CoSyNE single-hidden layer *Left*, CoSyNE multi hidden-layers *Center* and NEAT *Right* evaluation functions architectures over generations. Differences between distributions that resulted to be significant after by Bonferroni multiple comparison correction are marked.