Neuroevolution for the Sustainable Evolution of Neural Networks

E. Otović[1,3], J. Lerga[1,3], D. Kalafatovic[2,3] and G. Mauša[1,3]
1 University of Rijeka, Faculty of Engineering, Department of Computer Engineering, Rijeka, Croatia
2 University of Rijeka, Department of Biotechnology, Rijeka, Croatia
3 University of Rijeka, Center for Artificial Intelligence and Cybersecurity, Rijeka, Croatia
eotovic@riteh.hr

***Abstract* - The predictive performance of a neural network depends on its weights and architecture. Optimizers based on gradient descent are most commonly used to optimize the weights, and grid search is utilized to find the most suitable architecture from the list of predefined architectures. On the other hand, neuroevolution offers a solution for the simultaneous growth of neural network architecture and the evolution of its weights. Thus, it is not limited by the user-defined list of possible architectures and can find configurations optimal for a specific task. Both approaches can be effectively parallelized and take advantage of modern multi-process systems. In this research, we compare neuroevolution and backpropagation in terms of the time consumed by the algorithm, the predictive performance of the neural network, and the complexity of the neural network. The total time for each algorithm is measured along with the times for each section of the algorithm and the time spent on synchronization due to the multi-process setting. The neural networks are compared by their predictive performance in terms of Matthews correlation coefficient score and their complexity as the number of nodes and connections. The case study is based on two synthetic and two real-world datasets for classification tasks.**

Keywords - machine learning; neuroevolution; neural networks; genetic algorithm; sustainability; classification

I. Introduction

Machine learning is a rapidly growing branch of computer science and is considered to be one of the enabling factors for industry 4.0. However, the powerful hardware required to run machine learning algorithms and its energy consumption have become a concern in terms of sustainability [1, 2]. Neuroevolution, a form of machine learning that evolves neural networks specific to a given task through an evolutionary process, may provide a more sustainable alternative to traditional gradient-based machine learning methods.

Gradient-based optimization algorithms have become a *de facto* standard for training neural networks. These algorithms are generally computationally efficient and converge relatively fast, especially when datasets with a small number of features are provided. However, the design of neural network architecture and the choice of adequate hyperparameters poses an obstacle, especially when it comes to the application of machine learning to a problem that has not been previously studied [3]. Suggested configurations and practices from literature can serve as a good starting point, but ultimately, the near-optimal configuration is found through experimentation. Grid search is an automated way of searching for the best-performing hyperparameters configuration from the predefined set of possible hyperparameter values. It is a widely used method because it can be easily implemented and understood, but is also easily scaled to multiple processes for faster execution [3]. The search space for grid search is defined by the user and it quickly becomes inefficient when the search space is *too big*, as it grows rapidly with the number of dimensions and possible values. It may also end up with suboptimal solutions since it considers only the predefined values for each hyperparameter.

On the other hand, neuroevolution is a form of genetic algorithm which allows for the exploration of a large solution space which can lead to the discovery of novel and creative solutions and its search space is not bounded by user-defined configurations. Since it performs a global search, it is less likely to get stuck in local optima and is less dependent on initial conditions [4]. Finally, it yields neural networks that have the architecture as well as the weights adapted for the given task. Genetic algorithms are known for their high scalability and neuroevolution is not an exception. Being a gradient-free method, it allows the optimization of the neural network directly on the non-differentiable objective function. Because of this, it has gained popularity in reinforcement learning settings as it can directly aim to maximize the reward signal [5].

Neuroevolution removes the burden of user-defined search spaces and finds novel topologies, however, it is considered to converge slower than backpropagation-based neural network training. Optimizing the neural network topology using the backpropagation-based methods requires training and validation for every considered topology, while neuroevolution speeds up this process by optimizing weights and topology simultaneously. Hence, the goal of this study was to evaluate and compare neuroevolution and grid search in terms of execution time, model's predictive performance and complexity.

II. Materials and Methods

A. NeuroEvolution of Augmenting Topologies

Neuroevolution of augmenting topologies (NEAT) is a method first proposed by Kenneth Stanley in 2002 for evolving artificial neural networks through the use of genetic algorithms [6]. Its main distinction from traditional neural network training is that it allows for the topology to change during the evolution process. This can

lead to the discovery of novel topologies that are better suited for a specific task.

The algorithm flow diagram is shown in Fig. 1. It starts by initializing the random population of neural networks (solutions). The initial neural networks contain only the input and output nodes that are fully connected and the weights of connections are randomly initialized. Internally, it represents a neural network with a list of nodes and connections among nodes. Then, the population of initial solutions is evaluated. In the speciation step, the population is separated into multiple species based on genetic similarity with the aim to prevent premature convergence and maintain diversity. The main loop then starts and consists of three steps: reproduction, fitness evaluation and speciation. More fit individuals get to reproduce more often through natural selection and lead the population towards solutions that maximize the fitness function. The generated offspring form a population for the next generation. The algorithm runs until a stopping criterion is met, then exits the loop and returns the fittest individual as a solution to the problem.

The next generation is formed through elitism, crossover and mutation operators. In this paper, we employ species-wise elitism which preserves two best-performing individuals from each species into the next generation and thus ensures that best-performing individuals are not lost during the evolution process. A crossover operator in neuroevolution involves combining genes (nodes, connections and weights) from two individuals to create an offspring. This allows for the exploration of new combinations of genes, potentially leading to improved solutions. We used three mutations like the ones introduced in the original NEAT implementation. *Add node mutation* disables and replaces a single existing connection in a network with a node and two connections; one going from the source to the newly inserted node and the other going from the inserted node to the destination. The weight of the incoming connection is 1, and the weight of the outgoing connection is set to the weight of the disabled connection. The activation function for a node is randomly sampled from a list of available activation functions. *Add connection mutation* randomly inserts a single connection between two previously unconnected nodes and randomly initializes its weight. Since this study was focused only on feedforward neural networks, only connections that do not form a cycle in a network are allowed. *Weight mutation* randomly changes a fraction of connection weights in a network, either by adding a random value to them or by replacing them with a random value. The probability for each mutation is configured separately. The probabilities for *add node mutation* and *add connection mutation* control the rate at which new topologies are explored. If they are too low, the exploration will be slow and it will slow down the convergence of the algorithm. On the other hand, if they are too high, individuals may not have enough time to get their weights optimized through crossover and *weight mutation*. The fraction of offspring generated only by mutation is controlled by the *crossover probability* parameter.

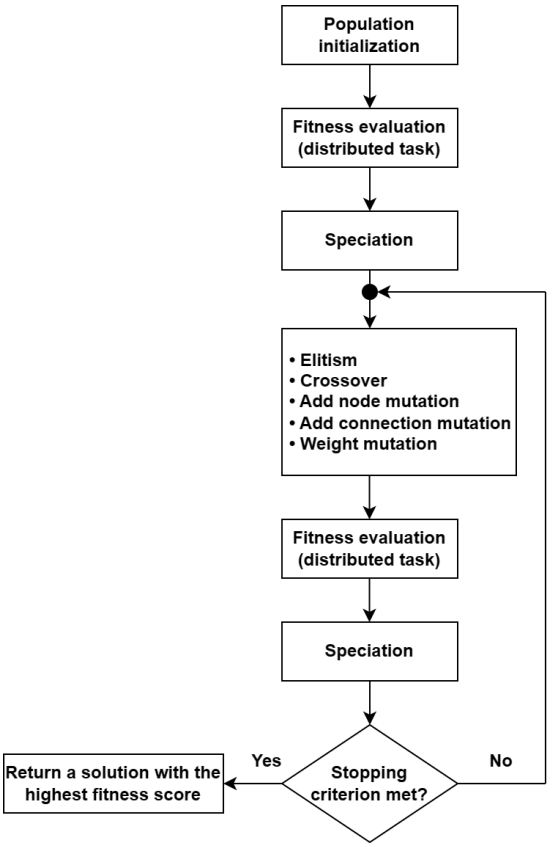

Figure 1. Program flow of NEAT.

NEAT aims to prevent premature convergence and preserve diversity in a population through speciation. Individuals from the population are grouped into species based on their genetic similarity and the best-performing individual from each species is chosen as its representative. When a newly formed individual has to be assigned to a species, it is sequentially compared against the representative from each species. The unnormalized compatibility distance **δ** shown in (1) is used in the comparison and if it is below a certain threshold $\boldsymbol{\delta}_t$ then the individual is assigned to that species.

$$\delta = c_{disjoint} * D + c_{weights} * \overline{W} \qquad (1)$$

The compatibility distance computes the genetic similarity between individual and species representatives. Its lowest value is zero when two genomes are equal, while higher values indicate greater difference. It considers the number of disjoint genes D (nodes and connections), which represents the number of genes that are present only in one of the individuals, but not both. It also considers the average difference in connection weights $\overline{W}$ for connections that are common to both individuals. The contribution of genes and weights to the compatibility distance can be regulated with $c_{disjoint}$ and $c_{weights}$ parameters. NEAT effectively tracks genes by assigning them historical markings that make it easy to determine genes that are shared by two individuals. Speciation allows the algorithm to explore solutions around multiple maxima at the same time. Members of

TABLE I. OVERVIEW OF USED DATASETS AND THEIR PROPERTIES.

Dataset name	Type	Number of features	Number of instances per class	Total size
Moons	Synthetic	2	10000, 10000	20000
Circles	Synthetic	2	5000, 5000	10000
Iris	Real-world	4	50, 50, 50	150
Breast cancer	Real-world	30	212, 357	569

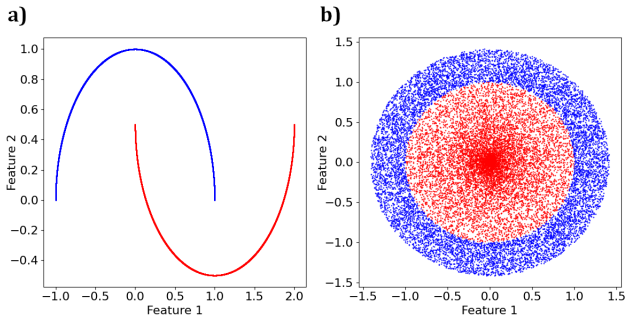

Figure 2. Visualization of moons (a) and circles (b) datasets.

the same species reproduce among themselves and on rare occasions interspecies mating occurs. The species that have not increased their fitness for a certain number of generations become stagnant and are not allowed to reproduce.

Fitness sharing is also introduced to prevent a single species from taking over the entire population. Equation (2) demonstrates the calculation of the shared (modified) fitness *f'* for an individual with fitness *f* that belongs to the species of size *n*.

$$f' = \frac{f}{n} \tag{2}$$

B. Algorithm and Model Evaluation

In this study, the algorithms and models are evaluated in terms of execution time, predictive performance of the model and model's complexity.

The total execution time is measured for both algorithms. In the NEAT algorithm, we as well measure the time required to complete the fitness function evaluation and the time spent on speciation and reproduction.

We used the Matthews correlation coefficient (MCC) to assess the predictive performance of the models. For the binary classification problems, it is defined as in (3):

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{3}$$

Variables *tp*, *tn*, *fp* and *fn* denote the number of true positives, true negatives, false positives and false negatives, respectively. Even though a confusion matrix cannot be described perfectly with a single number, the MCC score is regarded as one of the best options since it considers all four cells of the confusion matrix in the case of binary classification. This also makes it suitable for datasets with imbalances among the classes. MCC score has been generalized and can also be used in multi-class and multi-label classification problems [7].

The Wilcoxon signed-rank test is a non-parametric statistical test used to compare paired samples. In this case, it was used to compare the MCC score of neuroevolution models and grid search models in order to determine whether one of two algorithms performs significantly better than the other one. We used a significance level of 0.05.

The complexity of a neural network can be expressed as the number of nodes and connections between them. For the purpose of this paper, this count does not include the nodes in the input and output layers because we are interested only in the number of hidden nodes that are required to solve the task. We count each bias term as a single connection.

III. CASE STUDY

A. Datasets

In this case study we employ two synthetic and two real-world classification datasets. Both synthetic datasets have two features and two equally distributed classes. These datasets contain 2D points belonging to two objects; in the case of the *moons* dataset it is two curves resembling the shape of the moons (Fig. 2a), and in the case of the *circles* dataset it is a circle with a ring around it (Fig. 2b). Two input features represent *(x, y)* coordinates of the point while the task of the model is to predict which object the point belongs to.

The *iris* dataset contains instances of three different species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor) with four features representing various aspects of the flowers (sepal length, sepal width, petal length and petal width). Breast Cancer Wisconsin (Diagnostic) dataset contains malign and benign instances of breast cancer. Thirty provided features describe the characteristics of the cell nuclei present in the digitized image of a fine needle aspirate (FNA) of a breast mass. *Moons*, *Iris* and *Breast cancer* datasets were used from the *scikit-learn* library, while *circles* dataset was manually created [8, 9]. The overview of the used datasets is given in Table I.

Features used to describe instances in a dataset can have different units and scales which poses a problem for neural networks because the ones with a greater range of values may have a greater impact on the prediction outcome. To combat this, features were standardized for NEAT and backpropagation algorithms by centering the data points around zero and scaling to the unit variance [10]. If observed values for feature *i* are denoted as x_i, then standardization of that feature can be expressed as in (4).

$$x_i' = \frac{x_i - mean(x_i)}{std(x_i)} \tag{4}$$

To accurately estimate an algorithm's ability to adapt and model's ability to predict new data, we use stratified 10-fold cross-validation. The stratification guarantees that the distribution among classes will be preserved and thus prevents the under-representation of the minority class. In each iteration, nine folds are designated for training and one fold is held out for testing. In this way, each instance will be used exactly once for testing and thus minimizing the impact of random sampling on the performance estimation.

B. Algorithm Parameters

We used the NEAT algorithm with a population size of 250 and it was run for a maximum of 1000 generations or was stopped earlier if the maximal fitness did not increase for 300 generations. Output neurons used the softmax activation function while ReLU was used for hidden nodes. Weights were randomly sampled from a normal distribution with a mean of 0 and a standard deviation of 0.4. The crossover probability was set to 0.8, and the probability for interspecies mating was set to 0.01. *Add node mutation* probability was set to 0.1, *add connection mutation* was set to 0.3 and *weight mutation* was set to mutate 40% of all weights. A dynamic compatibility threshold was employed to preserve the number of species around 10 and the species would become stagnant after 20 generations without improvement. Compatibility parameters $c_{disjoint}$ and $c_{weights}$ were set to 2 and 1, respectively.

$$f = MCC + 1.001 \quad (5)$$

Equation (5) shows the fitness function that was used for neuroevolution. The original NEAT implementation cannot handle negative fitness scores due to its implementation of fitness sharing which is a problem since the lowest possible value of MCC is -1. Therefore, we had to increment it by 1.001 which allowed us to use it as a strictly positive fitness function.

Grid search tested three values for ADAM optimizer learning rate (0.01, 0.001, 0.0001) while other parameters were kept at Keras default values. We also varied the number of hidden layers (1, 2, 3), the number of neurons in each of them (10, 30, 50, 70, 90, 110) and dropout (0, 0.1, 0.2, 0.3, 0.4, 0.5). In total, this yields 4644 configurations that had to be tested. All the hidden layers used ReLU as the activation function. The output layer used the softmax activation function and the number of neurons corresponded to the number of classes in a given dataset. We used the Adam optimizer with a batch size of 32. Early stopping and dropout on each hidden layer were used to prevent overfitting. The training proceeded for the maximum of 20 epochs for *moons* and *circles* datasets due to their relatively large size, while the maximal number of epochs for *iris* and *breast cancer* datasets was set to 300 which provided enough time for convergence. 30% of the training was kept aside as the validation set and was used to evaluate and pick the best performing neural network configuration after training. The one with the highest MCC was evaluated on a held-out test set and that score is reported in Table II.

TABLE II. AVERAGE VALUES OF PREDICTIVE PERFORMANCE AND COMPLEXITY OF NEURAL NETWORKS FOUND BY NEAT AND GRID SEARCH. THE NUMBER OF EVALUATED NETWORKS, EXECUTION TIME AND THE NUMBER OF GENERATIONS ARE ALSO SHOWN. THE ASTERISK INDICATES STATISTICAL SIGNIFICANCE AT THE CHOSEN LEVEL OF SIGNIFICANCE (α=0.05).

	NEAT				**Grid search**			
	Moons	**Circles**	**Iris**	**Breast cancer**	**Moons**	**Circles**	**Iris**	**Breast cancer**
MCC score (test set)	1.0	0.913*	0.972	0.974*	1.0	0.981*	0.922	0.941*
Number of hidden nodes	8.8	14.9	7.5	8.2	10.0	229.0	69.0	89.0
Number of connections	39.3	69.9	39.6	87.6	52.0	12 717.0	2 578.0	3 597.0
Generations count	506	962	487	617	/	/	/	/
Evaluations count	99 038	192 412	95 378	122 940	4 644	4 644	4 644	4 644
Total time (per fold)	2 153s	4 135s	345s	610s	7 197s	6 447s	2 088s	2 261s
Evaluations per second	46.0	46.53	276.46	201.54	0.65	0.72	2.22	2.05

C. Experimental Setup

The experiments were carried out on a single computer with two Xeon E5 processors (24 physical cores; 48 threads) with 64 GB of RAM. Both algorithms were run in parallel with 12 workers. In the case of NEAT, multiple workers were used for parallel evaluation of genomes and in the case if grid search, it was used for parallel evaluation of neural network configurations. Due to the relatively large memory footprint of Tensorflow, it was not possible to use more workers on the same machine. Parallel execution of Python code was achieved with the *Dask* library which enables parallel execution on a single or across multiple computers. The dataset was distributed to all workers prior to the execution of the experiment. During each fitness evaluation in NEAT, genomes from the population had to be scattered among the workers, while in the case of the grid search, only numerical values describing neural network configuration and hyperparameters had to be sent to the workers. The fitness function or specific neural network configuration is evaluated in a worker process and the result is returned to the main process.

IV. Results and Discussion

Results from Table II show that NEAT completed on average 3.7 times faster than grid search. Even though our experiment was set up in a way that would allow both algorithms to find relatively good solutions in a reasonable time, it is impossible to fairly compare their performances in terms of total execution time. This is because the duration of the execution of each algorithm primarily depends on the configuration set by the user, e.g. stopping criteria for NEAT and search space size for a grid search. Grid search evaluated 4 644 pre-set neural network configurations for each dataset, while the number of neural networks evaluated by NEAT depends on many (stochastic) factors and cannot be known in advance. Therefore, we considered the number of neural networks evaluated per second which reflects the exploration rate of each algorithm. We found that NEAT is capable of performing exploration 90 times faster than the grid search. Grid search spends most of its time in the training phase tuning the weights of neural networks while the evaluation phase is relatively cheap time-wise. On the other hand, NEAT is capable of creating a new candidate neural network and evaluating it much more quickly. If the topology is predefined and fixed, backpropagation is an efficient algorithm to fine-tune the weights. Some researchers have proposed their combination in which NEAT would evolve the topology while backpropagation is used to fully or partially tune the weights [11, 12]. However, such methodology may not be applicable in some cases, e.g. if a non-derivable activation function is used.

Results from Table III show that NEAT spends most of the time evaluating the fitness function (on average 73.5%) even with parallelization introduced. This is expected because it involves the construction of a neural network from genes in a genome and performing forward pass to compute the output. The former depends on the size of the neural network while the latter depends on the size of the neural network and the size of the dataset. It is noticeable that larger datasets (*moons* and *circles*) required more time for forward pass than smaller datasets (*iris* and *breast cancer*). The second most intensive step is reproduction which consists of elitism, crossover and mutation and takes 19% of the time. The most expensive operations from this step are the identification of joint and disjoint genes during the crossover, verification if a new connection can be inserted without creating a cycle during *add connection mutation* and copying large genomes during elitism and crossover. The third most time-consuming step is speciation which computes the genetic distance between the genome and the representative of each species and takes 6.8% of the time. This involves finding the number of joint and disjoint genes as well as the average distance in connection weights which makes this computation relatively expensive. All other steps such as the computation of shared fitness are relatively fast and take on average 0.8% of the time. Therefore, fitness evaluation, reproduction and speciation are all dependent on the size of the genome and as evolution progresses and genomes get bigger, these steps require more time to complete.

Comparing the performance of the model evolved by neuroevolution and the model found by grid search, it is evident that both of them have achieved high MCC scores for all datasets. Both algorithms have achieved the highest possible score on the *moons* dataset which can be attributed to its relatively simple *shape* and the fact that classes are well separated (Fig. 2a). Statistical test revealed that grid search outperformed NEAT on *circles* dataset by a statistically significant margin of 7.4%. NEAT outperformed grid search by 5.4% and 3.5% on the *iris* and *breast cancer* datasets, respectively, which are relatively small in comparison to the other two datasets. While a statistically significant difference was found only in the case of the *breast cancer* dataset, it is worth noting that NEAT was significantly faster than grid search in every case, highlighting its potential as a more efficient approach. These results may suggest that NEAT is more suitable for smaller datasets.

In this research, we employed early stopping and dropout to control the overfitting. Early stopping mechanism evaluates model performance (generalization power) on a portion of the training set that was not used for training and stops training if validation loss stagnates

Table III. Measurements of Time NEAT Spent in Each Part of the Algorithm During a Single Generation. Measurements are Reported in Milliseconds.

Function	**Moons**	**Circles**	**Iris**	**Breast cancer**
Fitness evaluation	4 020 **(92%)**	4 003 **(91%)**	457 **(55%)**	582 **(56%)**
Speciation	77 **(2%)**	91 **(2%)**	87 **(10%)**	133 **(13%)**
Reproduction time	284 **(6%)**	298 **(7%)**	282 **(34%)**	298 **(29%)**
Other	8 **(<1%)**	12 **(<1%)**	9 **(1%)**	21 **(2%)**
Total	**4 389**	**4 404**	**835**	**1 034**

for 10 epochs. We tested multiple values for dropout which increased the search space by a factor of 6. When early stopping criteria is met, training is stopped and weights are restored to the ones with the lowest validation loss. Hyperparameters that are commonly used for addressing known problems can be found in the literature. However, for new or unfamiliar problems, these parameters need to be determined either manually or through an automated approach such as grid search. This also introduces the problem of choosing the amount of data that should be kept aside for early stopping. The more data is kept aside, the better the estimate of the model's performance will be, but at the same time less data will be available for training. On the other hand, setting too little data aside will result with high variance in the estimation of generalization by early stopping and may stop training too soon or too late. Therefore, it is a trade-off between the precision of the generalization power estimate and the amount of information that can be extracted from the training set.

The results from Table II reveal that neural networks evolved by NEAT are consistently less complex with regard to the number of hidden nodes and connections and this can be attributed to two factors. The limiting factor for traditional machine learning is that layers are stacked upon each other and fully connected. Even though this increases the model capacity, it introduces unnecessary connections and makes the neural network more prone to overfitting. On the other hand, NEAT starts with the population of minimal fully connected neural networks and the complexity of candidate topologies gradually increases through trial and error exploration. The evolutionary process of NEAT prevents the addition of unnecessary connections by favouring the solutions with higher fitness. Therefore, if a newly added node or a connection does not contribute to the fitness, eventually it will cease to exist. Additionally, using the historical markings, NEAT keeps track of previously inserted nodes and connections and utilizes it to prevent duplicate nodes and connections to be inserted. Neuroevolution operates on a finer level, e.g. activation function is separately set for each neuron and custom connections between nodes. Grid-search can be modified to consider more specialized architectures to some extent, e.g. by trying multiple activation functions for each layer, but such configurations rapidly increase the search space and are still inferior to NEAT in terms of customizability.

V. Conclusion

In this paper, we compared the NEAT neuroevolution algorithm and grid search in terms of execution time, predictive performance and neural network complexity. The measurements of execution time have shown that NEAT is capable of exploring the search space 90 times faster than grid search. However, in some cases, it is possible to combine neuroevolution and backpropagation to preserve the exploration of topologies and efficient weight-tuning mechanisms. With the addition of more parameters to the grid search, better results may be achieved at the cost of execution time. We demonstrated the ability of neuroevolution to more effectively search a much larger search space of possible configurations that are specialized for the given task. This resulted in NEAT consistently evolving neural networks with fewer nodes and connections than the ones that were found by grid search. In future studies, it would be valuable to compare NEAT against other commonly used hyperparameter optimization techniques, such as random search or Bayesian optimization, in order to gain a deeper understanding of the relative strengths and weaknesses of these methods and their applicability to different neural network optimization problems. We also plan to explore the potential of NEAT to improve the accuracy and efficiency of predictive models in peptide chemistry where a high number of available representation schemes together with high dimensionality and imbalance of data are an important challenge that requires smart optimization procedures. Our research findings suggest that NEAT is a promising approach for improving peptide activity prediction, particularly in cases where datasets are limited in size. Simpler neural networks are more suitable for deployment on battery-powered embedded devices in order to reduce computational demands and energy consumption while improving the responsiveness of the system to the input. This can lead to more sustainable and eco-friendly applications of artificial intelligence in emerging IoT devices.

Acknowledgment

This project was conducted using the resources of the supercomputer Bura at the University of Rijeka, Center for Advanced Computing and Modelling and supported by the Croatian Science Foundation/Hrvatska zaklada za znanost (grant no: UIP-2019-04-7999 and DOK-2020-01-4659).

This paper acknowledges the support of the Erasmus+ Key Action 2 (Strategic partnership for higher education) project No. 2020-1-PT01-KA203-078646: "SusTrainable - Promoting Sustainability as a Fundamental Driver in Software Development Training and Education". The information and views set out in this paper are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

References

[1] Peres, Ricardo Silva, et al. "Industrial artificial intelligence in industry 4.0-systematic review, challenges and outlook." IEEE Access 8 (2020): 220121-220139.

[2] "The Fourth Industrial Revolution: what it means, how to respond", World Economic Forum, Jan. 2016, https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/

[3] Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." Neurocomputing 415 (2020): 295-316.

[4] Kumar, Manoj, et al. "Genetic algorithm: Review and application." Available at SSRN 3529843 (2010).

[5] Igel, Christian. "Neuroevolution for reinforcement learning using evolution strategies." The 2003 Congress on Evolutionary Computation, 2003. CEC'03.. Vol. 4. IEEE, 2003.

[6] Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." Evolutionary computation 10.2 (2002): 99-127.

[7] Gorodkin, Jan. "Comparing two K-category assignments by a K-category correlation coefficient." Computational biology and chemistry 28.5-6 (2004): 367-374.

[8] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[9] Newman, David J., et al. "UCI repository of machine learning databases, 1998." (1998).

[10] Kotsiantis, Sotiris B., Dimitris Kanellopoulos, and Panagiotis E. Pintelas. "Data preprocessing for supervised learning." International journal of computer science 1.2 (2006): 111-117.

[11] Chandra, Rohitash, and Christian W. Omlin. "The Comparison and Combination of Genetic and Gradient Descent Learning in Recurrent Neural Networks: An Application to Speech Phoneme Classification." Artificial Intelligence and Pattern Recognition. 2007.

[12] Cui, Xiaodong, et al. "Evolutionary stochastic gradient descent for optimization of deep neural networks." Advances in neural information processing systems 31 (2018).