Large-Scale Evolutionary Optimization of Artificial Neural Networks Using Adaptive Mutations

Rune Krauss
DFKI
Bremen, Germany
rune.krauss@dfki.de

Jan Zielasko
DFKI
Bremen, Germany
jan.zielasko@dfki.de

Rolf Drechsler
University of Bremen / DFKI
Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—Big data forms the basis for training and testing models in *Machine Learning* (ML). The more data are available, the more effectively and accurately ML models can process them. Interest in ML has increased significantly as datasets are growing rapidly due to technological progress. One of the most popular ML models is an *Artificial Neural Network* (ANN), which is used in numerous areas such as medicine and the video game industry to analyze data and make decisions accordingly. In applications such as automatic speech recognition, sufficient labeled samples are provided to use supervised learning. Other applications such as robot control define an overall target while the expected output for a given input can be ambiguous, making supervised learning approaches not applicable. Neuroevolution, a kind of artificial intelligence that uses genetic algorithms for evolving ANNs, has been created to address this type of problem. However, one of the main issues is its mutation operator because it works randomly in general. This makes the exploration for performant ANNs more difficult or can easily disrupt feasible solutions. To tackle the aforementioned issue, this paper proposes adaptive mutations for ANN optimization. Experiments on several large-scale benchmarks show that the proposed approach evolves efficient ANNs and clearly outperforms related work.**

***Index Terms*—Evolutionary optimization, adaptive mutation, artificial neural networks, large-scale problems**

I. Introduction

Big data refers to complex datasets that can be generated by Internet of Things devices such as smartphones [1]. Collected data are stored, i. a., in graph databases [2] that can be analyzed by companies using frameworks such as Apache Spark™ [3] to predict customer demand and develop new products [4]. Applications include, e. g., image recognition that infers visible objects based on several complex rules [5] and automatic speech recognition where words are identified from a dictionary on the basis of acoustic sequences [6].

Besides many advantages, big data also comes with numerous challenges. In addition to the fact that the amount of data doubles approximately every two years [7], unstructured and semi-structured data types such as video and audio still need to be curated [8]. While developments for their effective storage are considerably advanced [9], big data technologies for their efficient use are changing rapidly [10]. A promising enhancement in data processing to enable flexible scalability is *Machine Learning* (ML) [11].

This work was supported by the German Federal Ministry of Education and Research within the projects DI-OCDCpro (contract no. 16ME0938), FAIRe (contract no. 01IS23074), and ECXL (contract no. 01IW22002).

There is a need to continuously improve models and algorithms in ML to keep up with big data technologies [12]. A suitable model to implement artificial intelligences for big data problems is a deep *Artificial Neural Network* (ANN) consisting of neurons and weighted connections [13]. While ANNs designed for image recognition use pixels to distinguish objects [14], ANNs for automatic speech recognition receive acoustic observations for classification on a dictionary [15]. Gradient descent algorithms such as backpropagation usually adjust weights in such predetermined ANNs based on inputs and expected outputs to minimize a loss function [16]. In this way, input signals like pixels or phonemes can be mapped to an output signal like an object or a word with the aim of being able to generalize after supervised learning [17].

In addition to the described applications, especially (software) robot control tasks have attracted attention in recent years [18], [19]. These include deliveries of medical supplies with self-navigation technology [20] and improving the behavior of non-player characters [21]. Unlike supervised learning, unambiguous outputs do not typically exist for *Reinforcement Learning* (RL) problems [22]. With the help of algorithms such as deep Q-learning, a robot interacts with its environment and becomes more intelligent through feedback from value functions on its actions made utilizing backpropagation [23].

Another approach is taken by *Neuroevolution* (NE) that evolves ANNs using *Genetic Algorithms* (GAs) [24]. Compared to RL, direct exploration in the solution space is possible without indirect inference from value functions, and problems with hard-to-compute or without any gradients can be tackled effectively [25]. Studies have shown that NE is more performant than RL algorithms like Q-learning for several RL problems such as robot control [26], [27]. Unfortunately, the mutation used by NE to modify ANNs and explore the search space is known to be heavily random, which can either disrupt already found "good" solutions or require impractical effort to find acceptable solutions [28], [29], [30], [31], [32], [33].

To combat the above issue, we propose adaptive mutations, which re-activate neurons and re-use ANNs depending on the solution quality, with the main goal of increasing the ANN optimization efficiency. Experiments on large-scale RL problems demonstrate that our approach evolves efficient ANNs. They are on average around 38 % smaller compared to established NE algorithms and solve control tasks about 1.5 times faster.

In summary, the main contributions of this research work are listed as follows:

- Implementation of adaptive ANN mutations to efficiently explore huge search spaces for feasible solutions;
- Comparison of the proposed approach with established NE algorithms on large-scale RL benchmarks.

This paper is structured as follows: Section II summarizes ANNs and GAs as NE fundamentals in an attempt to keep this work self-contained. In Section III, an overview of related work is given and NE algorithms that have proven promising in the past are discussed. The mutations that adapt mutation probability according to the solution quality are proposed in Section IV. Section V describes the experiments conducted to evaluate the optimization efficiency of the proposed approach compared to random mutations. Finally, Section VI concludes the paper and discusses future work.

II. Preliminaries

A huge amount of various information is filtered daily by our brain. One example is pattern recognition where the human brain has to make important decisions quickly [5]. To mimic these real-world processes and solve ML problems from experience (data), ANN types such as feedforward and recurrent ANNs were created.

Definition 1. A feedforward ANN G is a tuple $(T, \Xi, S(0), \Phi)$ that consists of a topology, constraints, an initial state, and activation functions specified as follows:

The topology T is a pair (Ω, E) that consists of the frame Ω and connections E.

Ω is a tuple $(l_1, l_2, \ldots, l_C)$, where C is the number of layers and the number of neurons N_c per layer is denoted by $l_c := \{n_{c,i}\}_{i=1}^{N_c}$ $(1 \leq c \leq C)$. While l_1 means the input layer and l_C refers to the output layer, every other layer indicates a hidden layer.

$E := \{n'_{c,i} \to n_{m,j}\}$ consists of a set of relations from neurons $n'_{c,i}$ to destination neurons $n_{m,j}$ iff $m > c$, where $n'_{c,i} := \{n_{c,i}\} \subseteq l_c$ is a set of source neurons and $n_{m,j}$ $(1 \leq m \leq C, 1 \leq j \leq N_m)$ denotes the corresponding destination neurons to which those subsets are connected.

The constraints $\Xi := (\Xi_W, \Xi_B, \Xi_A)$ dictate value ranges for weights $\Xi_W \subset \mathbb{R}$, bias units $\Xi_B \subset \mathbb{R}$, and activities $\Xi_A \subset \mathbb{R}$ meaning the outputs of neurons.

$S(0) := \{W(0), B(0), A(0)\}$ is the initial state of G where $W(0) := E \to \Xi_W$, $B(0) := \Omega \to \Xi_B$, and $A(0) := \Omega \to \Xi_A$.

Each hidden and output neuron possesses an activation function $\phi(x) \in \Phi$ that transforms the received input x.

If G contains at least one hidden layer, it is called *deep*.

Remark. A recurrent ANN is similar to Definition 1, but with the difference that feedback loops are also provided: direct, indirect, lateral, or full feedback [13]. Recurrent ANNs are useful, e. g., for learning temporal dependencies in automatic speech recognition [15].

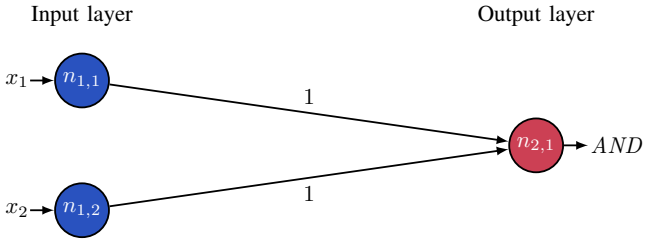


Fig. 1: ANN G representing the well-known Boolean function $AND(x_1, x_2) = x_1 \wedge x_2$, which is true iff its arguments are true. G consists of $N = 3$ neurons and $C = 2$ dense layers, meaning they are fully connected. The activation function ϕ corresponds to a threshold function that outputs 1 iff $x_1 + x_2 = 2$, 0 otherwise. Given an input combination, $n_{1,1}$ and $n_{1,2}$ transfer it to $n_{2,1}$ that calculates a certain activity level using ϕ and outputs a truth value accordingly. If the combination $(0, 1)$ applies, $0 \cdot 1 + 1 \cdot 1 = 1 \Longrightarrow 0$ is evaluated during forward pass. The other cases are analogous.

Algorithm 1 Standard GA for solving optimization problems

Input: Fitness threshold F_{th}
Output: Best individual

1: $t \leftarrow 0$ ▷ Generation number
2: Initialize population $P(t)$ consisting of μ individuals
3: **while** F_{th} is not reached **do**
4: Compute fitness $f(p)\, \forall p \in P(t)$
5: **for** $i \leftarrow 1, 2, \ldots, \mu$ **do**
6: Select parents $p_w, p_m \in P(t) : p_w \neq p_m$
7: $p' \leftarrow crossover(p_w, p_m)$
8: $mutate(p')$
9: $P(t+1)_i \leftarrow p'$
10: **end for**
11: $t \leftarrow t + 1$
12: **end while**
13: **return** best $p \in P(t-1)$

Example 1. The evaluation of an ANN that computes the logical conjunction is demonstrated in Fig. 1.

In practice, ML problems as in Example 1 are not linearly separable, so hidden layers are necessary [34]. With the help of a bias unit, threshold values of activation functions can be changed dynamically [13]. For these reasons, ANNs are theoretically capable of approximating any continuous function [35]. This property makes them powerful for robot control that is mainly about solution space exploration [19]. Since such RL tasks can be formulated as evolutionary optimization problems, GAs inspired by Darwinian natural selection and originally introduced by J. Holland [36] are suitable for finding feasible solutions [37]. Algorithm 1 shows a commonly used GA realization whose basics are explained in detail below.

The initial population $P(t)$ starts with μ random candidate solutions (individuals) in generation $t \leftarrow 0$, where each individual from $P(t)$ represents a point in the search space and μ is a predetermined hyperparameter (Lines 1–2). The objective

is to explore the search space for feasible solutions until the given fitness threshold F_{th} is reached (Line 3). To this end, a potential solution is encoded in a chromosome (genotype) whose quality evaluation is performed by a fitness function f to compute the respective loss and individual fitness (Line 4).

Example 2. Let $XOR(x_1, x_2) = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$ be a Boolean function that is true iff its arguments differ. Given the sigmoid activation function $\phi(x) = 1 \,/\, 1 + e^{-x}$ [13], the ANN loss is $f = 4 - \sum_i (e_i - a_i)^2$ between expected (e_i) and actual (a_i) outputs. If the ANN evaluates exactly to the expected outputs, f computes an individual fitness of 4, otherwise a score less than 4. The higher F_{th}, the more accurate the accepted solution.

Remark. It is rare that GAs are used in a supervised context as in Example 2 because there is usually enough gradient information for gradient descent algorithms to work well.

In order to change genotypes and thereby explore the solution space, individuals are successively placed into a so-called *mating pool* using fitness proportionate selection [37], meaning those with higher fitness scores have a higher probability of reproducing (Lines 5–6). While the genetic operator $crossover$ mixes parental genes in the hope of breeding fitter offspring (Line 7), $mutate$ causes random perturbations to a genotype (Line 8). Analogous to biological mutations, it ensures genetic diversity from one generation to the next (Lines 9–12) [38]. Finally, the individual with the highest fitness score is returned (Line 13).

In summary, if the genotype corresponds to an ANN and a GA is used to "train" ANNs gradient-free by applying crossover and mutation, this is referred to as NE.

III. Related Work

Biologically, neurons are interconnected, but their communication is not just static [39]. Conventional NE only evolves connection weights and often fails to converge if predetermined ANNs are not deep enough to solve a specific RL problem [26]. Many enhancements have been researched in NE for this reason, especially in terms of *Topology and Weight Evolving ANN* (TWEANN), as such algorithms can also evolve neurons and connections. A comprehensive survey is available in [30], [32]. TWEANN algorithms that have achieved promising results are briefly discussed below.

The most famous TWEANN algorithm for evolving ANNs from scratch is probably *NeuroEvolution of Augmenting Topologies* (NEAT) [28] that has basically the following technical strengths: direct genetic encoding and speciation. To tackle the competing conventions problem [40], each connection contains a historical marking. This makes ANNs unique so that their crossover cannot produce damaged offspring. Furthermore, these markings are used to efficiently compare individuals based on their ANN topology in order to speciate them. They compete in their own niche, which counteracts the individual dominance problem [41].

Turbo NEAT [29] extends NEAT with a divide-and-conquer concept. Its general idea is to use multiple populations for training ANN combinations. Populations are evolved simultaneously, i. e., every ANN from each population is evaluated in combination with every other ANN from the other populations. This approach can also be configured statically so that samples are divided into sequences for each of which a population is responsible. Samples for non-sequential problems (Example 2) can additionally be clustered to evolve multiple ANNs whose weights are subsequently refined using backpropagation.

Artificial Life Form (ALF) [31] is an evolutionary framework with which a TWEANN algorithm was implemented that is oriented towards Turbo NEAT but offers remarkable benefits: semantic speciation, dynamic populations, and fitness-based genetic operations. First, besides ANN topology, outputs are also used for speciation to improve evolutionary exploration of solution spaces. Second, the population is resized depending on extinct species in order to leave unsuitable subspaces more quickly. Third, fitness is utilized to increase the probability of optimization success, meaning that the higher the score, the fewer ANN mutations occur, and vice versa. These include adding layers, hidden neurons, bias units, and connections as well as weight adjustments. Since ALF's genotype is generic, it can be used in many fields for various purposes such as binary decision diagram optimization [42].

To address the lack of scalability, genetic recombination for efficient mixing of entire ANNs and ensemble learning methods [43] to restrict huge solution spaces to be searched exist in the TWEANN algorithm implemented using ALF [33]. For the sake of simplicity, this algorithm is referred to directly as ALF in the following. While the so-called *Output Distribution* method assigns previously identified subproblems to different genotypes of each individual for evolving specialized ANNs, bootstrap aggregating trains multiple weak learners that collaboratively predict for ANN inputs.

Although ALF has been shown on benchmarks to interact more intelligently in non-deterministic environments and to have higher generalization performance due to reduced overfitting compared to state of the art, its mutation probability is generally assigned a constant value. Even though, unlike related TWEANN algorithms, fitness is considered as a mutation parameter, it essentially only controls the number of mutation attempts. The following scenarios can therefore easily disrupt learning progress or lead to stagnation:

1) Evolving dead neurons;
2) Re-use of unsuitable ANN structures.

These limitations are to be overcome in order to meet the main goal of this work as stated in Section I.

IV. Proposed Adaptive Mutations

To tackle the issues identified in the previous section, they are addressed by our novel adaptive mutations that choose a mutation probability depending on individual fitness:

1) Living Neuron (Section IV-A);
2) Structural Learning (Section IV-B).

The listed strategies learn useful ANN mutations while the corresponding mutation probability refers to ANN regions, which is proposed at the beginning.

(a) $\alpha = 1.0$, $\beta = 1.0$ (b) $\alpha = 0.3$, $\beta = 1.0$ (c) $\alpha = 1.0$, $\beta = 3.0$

Fig. 2: Probability matrices w. r. t. Eq. (1), where the rows refer to ANN layers and columns mean the (inverse) fitness scores. The lighter a cell is, the more likely a mutation in the layer, and vice versa. No mutation would thus occur for an accepted solution. In (a), a linear relationship of layers, and scores is visualized. The other matrices interpolate differently. While (b) strengthens mutations in anterior ANN regions, (c) exerts more influence on individuals in early stages.

Anterior ANN regions are more likely to have more influence on ANN behavior than posterior regions [34]. When applied to TWEANN, it is less likely that a mutation in anterior layers will improve ANN performance the closer an individual is to the fitness threshold. On the contrary, there is an increased risk that their solution quality will deteriorate. Since ALF essentially only uses fitness quantitatively in mutations, even one ANN modification can destroy the learning progress of a fit individual. The population would stagnate if features like elitism [41] are enabled.

To avoid the pits of total randomness of mutation, a probability distribution for ANN layers according to the fitness score is introduced, which is shown in Eq. (1).

$$\rho(C_l, F) = \left(\frac{C_l}{C}\right)^{\alpha} \cdot \left(1 - \frac{F}{F_{th}}\right)^{\beta} \tag{1}$$

$C_l \in \mathbb{N}$: ANN layer ID
C : Number of layers
α : Layer scaling
F : Individual fitness
F_{th} : Fitness threshold
β : Fitness scaling

In general, the fitter an individual is, the less likely their ANN changes in anterior layers during mutation, and vice versa. To allow flexibility, the respective mutation probabilities can be shifted using the exponents, which is illustrated in Fig. 2.

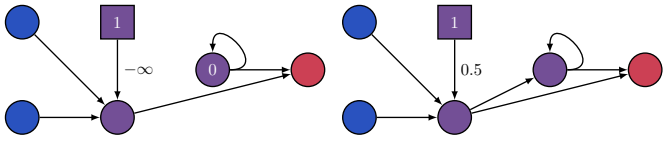
Specifically, Eq. (1) is utilized by the novel strategies that specify ANN changes based on it, which are proposed next.

A. Living Neuron

Dead neurons are a type of vanishing gradient problem [13] where the gradient of the loss function w. r. t. layer weights becomes very small or zero, meaning weights are not updated during training and the respective layer becomes inactive.

Definition 2. Let G be an ANN according to Definition 1 that also provides feedback. A neuron in G is called *dead* if it produces an unchangeable activation regardless of its input.

Dead neurons commonly occur in deep ANNs that use activation functions with limited ranges, such as sigmoid (Example 2) or ReLU $\phi(x) = \max(0, x)$ [34]. For example,



(a) Random (b) Adaptive

Fig. 3: Comparison between a randomly mutated ANN and an adaptively mutated ANN. In (a), the ANN is disconnected because the neuron labeled by 0 has no path to the input layer. The other hidden neuron is also dead if ReLU is used since the connection of the bias unit visualized by a square shape has a high negative weight multiplied by 1. In (b), this weight is reinitialized to 0.5. If the corresponding individual was fit, the negative bias change in this anterior layer would have been less likely. The former neuron can live depending on its inputs, as it is on a path from the input to the output layer.

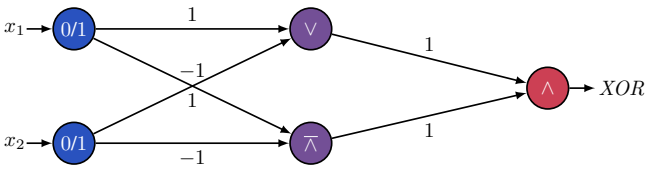


Fig. 4: ANN with one hidden layer consisting of two neurons to separate the XOR problem (Example 2). The Boolean disjunction operation $\vee$ outputs 1 iff its input is positive. The threshold x of the negated conjunction $\overline{\wedge}$ is $x > -2$. The $\wedge$-neuron corresponds to the ANN in Fig. 1.

ReLU neurons can be fragile during training and die when input ranges are strongly negative. If there are too many dead neurons, the ANN loses much of its explanatory power since many neurons do not contribute to the ANN output [13]. This state is established more easily in TWEANN or ALF because a disconnected ANN as shown in Fig. 3(a) can also emerge. The worst-case scenario, where the entire ANN dies and just a constant function is computed, is also possible.

To prevent dead neurons due to missing connections, a path rule is defined: Whenever a neuron is evolved in ALF by a mutation, it must be on a path between the input and output layer. Activation values are monitored during the evolution and testing of ANNs to detect dead neurons. The memory overhead caused by this procedure is negligible because comparison values can be overwritten after each generation. Made records are utilized to no longer mutate outgoing connections from dead neurons. Instead, their incoming connections are reinitialized so that they can become useful (again), which is shown schematically in Fig. 3(b).

B. Structural Learning

McCulloch-Pitts neurons [44] are part of many of today's ANN types. These neurons form a complete base for Boolean functions since they can simulate logical conjunction (Example 1), disjunction, and negation [13].

Example 3. Fig. 4 depicts an ANN for exclusive alternation.

TABLE I: Experimental comparison between random and adaptive mutations in solving large-scale RL benchmark problems

Experiment		Random mutations						Proposed adaptive mutations					
Name	Fitness threshold	Learning time in min		Generation number		Number of neurons		Learning time in min		Generation number		Number of neurons	
		Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
DPNV	1,000 time steps	**2**	**1**	**9**	3	47	26	**2**	**1**	**9**	**2**	**22**	**8**
DPNV	1,500 time steps	6	**1**	16	7	45	21	**5**	**1**	**14**	**5**	**25**	**10**
DPNV	2,000 time steps	26	5	43	12	93	32	**20**	**2**	**36**	**7**	**68**	**21**
DPNV	2,500 time steps	54	16	71	19	146	46	**41**	**9**	**64**	**13**	**92**	**18**
DPNV	3,000 time steps	97	26	105	31	215	57	**69**	**12**	**88**	**17**	**144**	**33**
Snake	20 apples	13	5	21	4	96	30	**11**	**3**	**20**	**2**	**63**	**12**
SMB level 2-4	X-coordinate 2,167	629	246	206	78	361	173	**402**	**112**	**182**	**42**	**204**	**65**

Boolean operations play a central role in numerous RL applications such as robot control in order to make decisions depending on logical conditions [19], [20], [21], [22]. With the help of McCulloch-Pitts neurons, they can be easily encoded and combined as demonstrated in Example 3.

ALF is therefore being extended with the ability to mutate the McCulloch-Pitts model. The corresponding ANNs are learned in advance and stored statically in a vector. They are scaled during a mutation depending on the number of source neurons in order to re-use them structurally between a source and destination layer of an individual ANN. This mutation chooses both layers adaptively using Eq. (1) analogous to the mutation strategy presented in the previous subsection.

V. Experiments

This section summarizes the experiments conducted to empirically analyze our approach proposed in Section IV. To this end, Section V-A describes the system specification and RL problems used for the performance evaluation in Section V-B where the benefits of our adaptive mutations are demonstrated.

A. Setup

To evaluate the ANN optimization efficiency of the proposed approach, its adaptive mutations were implemented in ALF using C++23. The corresponding parameter values were set as shown in Fig. 2(a). To allow a fair comparison with random mutations included in the original ALF, the same system configuration as in [33] was used. Accordingly, evaluations were carried out on a Fedora 37 machine with an Intel® Xeon® E5-2680 processor and 32 GB of main memory. The hyperparameter settings were also adopted for the large-scale RL benchmarks considered: *Double Pole, No Velocities* (DPNV) [28], Snake [45], and *Super Mario Bros.* (SMB) [46]. For representative purposes, the divide-and-conquer method described in Section III was used for an experiment that proved most successful for robot control: bootstrap aggregating if DPNV, Output Distribution otherwise. Each experiment was repeated 10 times for complexity reasons.

B. Results

The experimental results are summarized in TABLE I, which are now interpreted and discussed. ALF with adaptive mutations evolves smaller ANNs for each experiment compared to random mutations, which solve every control task at least as quickly. Specifically, optimized ANNs of the best individuals are reduced by around 38 % on average, making the learning time about 1.5 times faster. For example, only about 2 instead of 3 min are needed to evaluate an entire SMB generation. In addition, the results are overall significantly more stable, as the standard deviation is roughly halved. The main reasons for these improvements are the dead neuron handling mechanism and re-use of ANNs according to individual fitness. While the former prevents fragile ANNs in particular, the latter provides an additional boost in learning combined actions. The fact that the novel mutations adapt their probability to overcome total randomness makes the learning process less prone to degradation or stagnation, which is particularly noticeable as individuals become continuously fitter. This progress leads to efficient ANNs that cannot only be evaluated rapidly, but also make meaningful predictions for inputs.

These results ultimately apply not only to learning, e. g., in non-deterministic environments such as Snake, but also to ML testing. In this context, the learned ANNs from SMB level 2-4 were used as in [33] and test runs were performed for levels 1-4 and 3-4. While the software robot in level 1-4 (3-4) originally only reached the x-coordinate 781 (720), it now stops at 1,094 (952), leading to an increase in generalization performance of approximately 36 %.

VI. Conclusion

In this paper, we presented adaptive mutations to address a lack of NE optimization. To this end, the focus was on dead neuron handling and structural ANN learning. Experiments on several large-scale benchmarks for robot control confirmed that the main goal of this work was achieved. Various RL problems can be solved considerably more efficiently using the proposed approach compared to random mutations. ANNs evolved by the novel mutations are on average about 1.5 times faster and around 38 % smaller in this context.

We believe that evolutionary ANN optimization can be further improved. Future research will therefore be primarily directed towards investigating and utilizing ANN structures. At present, learned ANNs can only be re-used statically. This strategy should be extended to also dynamically mutate evolved ANNs. Besides this mutation, convolutional neural network architectures for deep learning are also to be found in an evolutionary way. In addition, different hyperparameter settings will be analyzed using further robot control tasks.

REFERENCES

[1] A. Naghib, N. J. Navimipour, M. Hosseinzadeh, and A. Sharifi, "A comprehensive and systematic literature review on the big data management techniques in the Internet of Things," *Wireless Networks*, vol. 29, no. 3, pp. 1085–1144, 2022.

[2] M. Macak, M. Stovcik, and B. Buhnova, "The suitability of graph databases for big data analysis: A benchmark," in *Proceedings of the 5th International Conference on Internet of Things, Big Data and Security*. SciTePress – Science and Technology Publications, 2020, pp. 213–220.

[3] Apache Software Foundation, "Apache Spark™ – a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters," 2024. [Online]. Available: https://spark.apache.org

[4] D. Tosi, R. Kokaj, and M. Roccetti, "15 years of big data: A systematic literature review," *Journal of Big Data*, vol. 11, no. 1, pp. 1–39, 2024.

[5] S. Zhang, Y. Wu, and J. Chang, "Survey of image recognition algorithms," in *Proceedings of the 4th Information Technology, Networking, Electronic and Automation Control Conference*. IEEE, 2020, pp. 542–548.

[6] A. Rista and A. Kadriu, "Automatic speech recognition: A comprehensive survey," *SEEU Review*, vol. 15, no. 2, pp. 86–112, 2020.

[7] A. Stojanov and B. K. Daniel, "A decade of research into the application of big data and analytics in higher education: A systematic review of the literature," *Education and Information Technologies*, vol. 29, no. 5, pp. 5807–5831, 2024.

[8] M. Yang, "Research progress on data analysis in big data technology," in *International Conference on Computer Engineering, Information Science & Application Technology*. Atlantis Press, 2017, pp. 851–854.

[9] J. Akoka, I. Wattiau, and N. Laoufi, "Research on big data – a systematic mapping study," *Computer Standards & Interfaces*, vol. 54, no. 2, pp. 105–115, 2017.

[10] S. Leonelli, "Scientific research and big data," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Stanford: Metaphysics Research Lab, 2020, pp. 191–217.

[11] L. Wang and C. A. Alexander, "Machine learning in big data," *International Journal of Mathematical, Engineering and Management Sciences*, vol. 1, no. 2, pp. 52–61, 2016.

[12] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[13] K. Du and M. N. Swamy, *Neural Networks and Statistical Learning*. London: Springer, 2020.

[14] H. Wang, G. Li, Z. Ma, and X. Li, "Application of neural networks to image recognition of plant diseases," in *International Conference on Systems and Informatics*. IEEE, 2012, pp. 2159–2164.

[15] W. Gevaert, G. Tsenov, and V. Mladenov, "Neural networks used for speech recognition," *Journal of Automatic Control*, vol. 20, no. 1, pp. 1–7, 2010.

[16] C. Sekhar and P. S. Meghana, "A study on backpropagation in artificial neural networks," *Asia-Pacific Journal of Neural Networks and Its Applications*, vol. 4, no. 1, pp. 21–28, 2020.

[17] V. Nasteski, "An overview of the supervised machine learning methods," *Horizons*, vol. 4, pp. 51–62, 2017.

[18] D. K. Mishra, A. K. Upadhyay, and S. Sharma, "Role of big data analytics in manufacturing of intelligent robot," *Materials Today: Proceedings*, vol. 47, no. 19, pp. 6636–6638, 2021.

[19] V. K. Singh, S. Chen, M. Singhania, B. Nanavati, A. K. Kar, and A. Gupta, "How are reinforcement learning and deep learning algorithms used for big data based decision making in financial industries – a review and research agenda," *International Journal of Information Management Data Insights*, vol. 2, no. 2, pp. 1–15, 2022.

[20] X. Wang, Y. Li, and K. Kwok, "A survey for machine learning-based control of continuum robots," *Frontiers in Robotics and AI*, vol. 8, pp. 1–14, 2021.

[21] E. K. Sure and X. Wang, "A deep reinforcement learning agent for general video game AI framework games," in *International Conference on Artificial Intelligence and Computer Applications*. IEEE, 2022, pp. 182–186.

[22] J. Kober and J. Peters, *Learning Motor Skills: From Algorithms to Robot Experiments*. Cham: Springer, 2014.

[23] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proceedings of the SAI Intelligent Systems Conference*. Springer, 2018, pp. 426–440.

[24] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[25] R. Miikkulainen, "Evolving neural networks," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2016, pp. 229–253.

[26] K. O. Stanley and R. Miikkulainen, "Competitive coevolution through evolutionary complexification," *Journal of Artificial Intelligence Research*, vol. 21, no. 1, pp. 63–100, 2004.

[27] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1712.06567

[28] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[29] R. Krauss, M. Merten, M. Bockholt, S. Froehlich, and R. Drechsler, "Efficient machine learning through evolving combined deep neural networks," in *Proceedings of the GECCO Companion*. ACM, 2020, pp. 215–216.

[30] E. Papavasileiou, J. Cornelis, and B. Jansen, "A systematic literature review of the successors of "NeuroEvolution of Augmenting Topologies"," *Evolutionary Computation*, vol. 29, no. 1, pp. 1–73, 2021.

[31] R. Krauss, M. Merten, M. Bockholt, and R. Drechsler, "ALF: A fitness-based Artificial Life Form for evolving large-scale neural networks," in *Proceedings of the GECCO Companion*. ACM, 2021, pp. 225–226.

[32] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 6, pp. 476–493, 2021.

[33] M. Merten, R. Krauss, and R. Drechsler, "Scalable neuroevolution of ensemble learners," in *Proceedings of the GECCO Companion*. ACM, 2023, pp. 667–670.

[34] A. Bhaya, "Neural networks: Theory and practice," in *Decision Sciences*, R. N. Sengupta, A. Gupta, and J. Dutta, Eds. Boca Raton: CRC Press, 2016, pp. 751–800.

[35] G. Lewicki and G. Marino, "Approximation by superpositions of a sigmoidal function," *Journal of Analysis and Its Applications*, vol. 17, no. 10, pp. 1147–1152, 2004.

[36] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992.

[37] T. Alam, S. Qamar, A. Dixit, and M. Benaida, "Genetic algorithm: Reviews, implementations, and applications," *International Journal of Engineering Pedagogy*, vol. 12, no. 6, pp. 57–77, 2020.

[38] D. Whitley, "An overview of evolutionary algorithms: Practical issues and common pitfalls," *Information and Software Technology*, vol. 43, no. 14, pp. 817–831, 2001.

[39] J. Lin, "Artificial neural network related to biological neuron network: A review," *Advanced Studies in Medical Sciences*, vol. 5, no. 1, pp. 55–62, 2017.

[40] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE, 1992, pp. 1–37.

[41] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[42] W. Lenders and C. Baier, "Genetic algorithms for the variable ordering problem of binary decision diagrams," in *Foundations of Genetic Algorithms*, A. H. Wright, M. D. Vose, K. A. De Jong, and L. M. Schmitt, Eds. Berlin: Springer, 2005, pp. 1–20.

[43] C. Zhang and Y. Ma, *Ensemble Machine Learning*. New York: Springer, 2014.

[44] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[45] J. Yeh, P. Su, S. Huang, and T. Chiang, "Snake game AI: Movement rating functions and evolutionary algorithm-based optimization," in *Conference on Technologies and Applications of Artificial Intelligence*. IEEE, 2016, pp. 256–261.

[46] E. D. Demaine, G. Viglietta, and A. Williams, "Super Mario Bros. is harder/easier than we thought," in *Proceedings of the 8th International Conference on Fun with Algorithms*. Schloss Dagstuhl Publishing, 2016, pp. 1–14.