Explorations of the Semantic Learning Machine Neuroevolution Algorithm: Dynamic Training Data Use, Ensemble Construction Methods, and Deep Learning Perspectives

Ivo Gonçalves[1], Marta Seca[2], and Mauro Castelli[2]

[1] INESC Coimbra, DEEC, University of Coimbra, Coimbra, Portugal

[2] [c]NOVA IMS, Universidade Nova de Lisboa, Lisboa, Portugal
e-mail: M20170451@novaims.unl.pt - mcastelli@novaims.unl.pt

This is the Author Peer Reviewed version of the following chapter/ conference paper published by Springer:

Gonçalves, I., Seca, M., Castelli, M. (2020). Explorations of the Semantic Learning Machine Neuroevolution Algorithm: Dynamic Training Data Use, Ensemble Construction Methods, and Deep Learning Perspectives. In: Banzhaf, W., Goodman, E., Sheneman, L., Trujillo, L., Worzel, B. (eds) Genetic Programming Theory and Practice XVII. Genetic and Evolutionary Computation. Springer, Cham. https://doi.org/10.1007/978-3-030-39958-0_3

FUNDING:

This work was partially supported by projects UID/MULTI/00308/2019 and by the European Regional Development Fund through the COMPETE 2020 Programme, FCT—Portuguese Foundation for Science and Technology and Regional Operational Program of the Center Region (CENTRO2020) within project MAnAGER (POCI-01-0145-FEDER-028040). This work was also partially supported by national funds through FCT (Fundação para a Ciência e a Tecnologia) under project DSAIPA/DS/0022/2018 (GADgET).

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License**.**

Explorations of the Semantic Learning Machine Neuroevolution Algorithm: Dynamic Training Data Use, Ensemble Construction Methods, and Deep Learning Perspectives

Ivo Gonçalves, Marta Seca, and Mauro Castelli

Abstract The recently proposed Semantic Learning Machine (SLM) neuroevolution algorithm is able to construct Neural Networks (NNs) over unimodal error landscapes in any supervised learning problem where the error is measured as a distance to the known targets. This chapter studies how different methods of dynamically using the training data affect the resulting generalization of the SLM algorithm. Across four real-world binary classification datasets, SLM is shown to outperform the Multi-layer Perceptron, with statistical significance, after parameter tuning is performed in both algorithms. Furthermore, this chapter also studies how different ensemble constructions methods influence the resulting generalization. The results show that the stochastic nature of SLM already confers enough diversity to the ensembles such that Bagging and Boosting cannot improve upon a simple averaging ensemble construction method. Finally, some initial results with SLM and Convolutional NNs are presented and future Deep Learning perspectives are discussed.

Key words: Semantic Learning Machine, Neuroevolution, Evolutionary Machine Learning, Artificial Neural Networks, Deep Learning, Deep Semantic Learning Machine

Ivo Gonçalves
INESC Coimbra, DEEC, University of Coimbra, Pólo 2, 3030-290 Coimbra, Portugal, e-mail: ivogoncalves77@gmail.com

Marta Seca · Mauro Castelli
NOVA IMS, Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal, e-mail: {M20170451,mcastelli}@novaims.unl.pt

1 Introduction

The success of artificial intelligence can be partially attributed to Artificial Neural Networks (ANNs), a machine learning algorithm that was invented in the late 1950s [74]. Inspired by the anatomy of the human brain, classic ANNs consist of neurons: atomic operators that receive a set of inputs and generate one output, determined by their activation function. To create networks, neurons are connected over synapses, so that the output of one neuron serves as input for the other. ANNs were used for a wide variety of tasks in many different fields [62], showing their suitability in addressing both classification and regression problems. Several properties of ANNs make them a suitable approach for addressing forecasting tasks: (1) ANNs are data-driven self-adaptive methods and there are few a priori assumptions about the models for the problems under study [79]. Thus, ANNs are a valuable technique for problems whose solutions require knowledge that is difficult to specify, but for which there are enough data or observations; (2) ANNs are universal functional approximators. In particular, it was proven that ANNs can approximate any continuous function to any desired accuracy [31, 29, 28].

Despite this result, in the literature there is no established procedure to determine the correct number of neurons for a given application. Another critical aspect relates with the topology of the network, that is, how the neurons are connected among them. For ANNs to perform well at a certain task, it is critical to find suitable connection weights. For this purpose, weights are adjusted in a learning process, based on provided training data. The most prevalent approach is backpropagation [60], where the error between prediction and ground truth is distributed back recursively through adjacent connections. However, backpropagation fails to answer the question of how to define the general topology of neurons and synapses. Devising suitable topologies is crucial, since it directly affects the speed and accuracy of the subsequent learning process.

Neuroevolution addresses these issues by applying Evolutionary Computation (EC) methods with the goal of evolving ANNs. Within neuroevolution, some approaches are specifically designed to automatically discover suitable combinations of topology and weights. More recently, a neuroevolution algorithm called Semantic Learning Machine (SLM) was proposed by Gonçalves et al. [20]. The most interesting characteristic of SLM is that it searches over unimodal error landscapes in any supervised learning problem where the error is measured as a distance to the known targets. It was empirically verified that this characteristic allows SLM to outperform other neuroevolution methods over a considerable set of supervised learning problems [32]. This work continues the investigation of SLM by empirically studying: (1) different methods of dynamic training data use; (2) different ensemble constructions methods. Furthermore, the extension of SLM to Convolutional Neural Networks is also discussed.

This chapter is organized as follows: section 2 overviews the field of neuroevolution; section 3 describes the SLM neuroevolution algorithm and presents its distinctive features; section 4 outlines the experimental methodology; section 5 reports and discusses the experimental results; and section 6 presents some initial results with SLM and Convolutional Neural Networks, and discusses future Deep Learning perspectives.

2 Neuroevolution Overview

One of the most challenging problems when using an ANN is the choice of its architecture or topology. In this context, the terms architecture and topology are used as synonyms to indicate some specific hyperparameters of the ANN, namely the number of hidden layers, the number of hidden neurons, and how the neurons are connected among them. Despite the vast number of applications where ANNs were used [1], the literature lacks an established procedure to determine the most suitable topology of a network when addressing a given task. Consequently, a lot of effort is currently put into automating the process of finding good ANN architectures. Solving this requires addressing several topics, such as:

1. how to design the components of the architecture
2. how to put them together
3. how to set the hyperparameters

There are two main approaches followed when approaching these topics, namely a) using search methods based on artificial intelligence, and b) using evolutionary techniques to generate networks. The first approach uses the gradient descent algorithm to optimize the weights of the network and to dynamically modify the hyperparameters of ANNs, while the second approach is characterized by the use of evolution to optimize the network's topology.

Neuroevolution techniques were successfully applied in different domains [14] and several attempts were proposed for using EC techniques for the optimization of ANNs [76]. The existing works can be categorized into three main approaches: (1) use of EC to train the ANN; (2) use of EC to optimize the network of an ANN; (3) use of EC to optimize the ANN's topology and to train the ANN. These different approaches are briefly discussed to present the reader the evolution of this research field in the last three decades.

The initial works aimed at using EC techniques for optimizing the weights of the connections, with a fixed topology [53, 52, 73, 68, 10]. The main idea of these works was to counteract the limitations of the backpropagation algorithm [30, 7], that due to the use of gradient descent [71] can get trapped in a local minimum of the error function and it is not capable of determining a global minimum if the error function is multimodal and/or non-differentiable [76]. These works replace the backpropagation algorithm

with an evolutionary technique for learning the weights of the ANN's connections. The use of EC techniques for optimizing the weights of an ANN is attractive because they can handle the global search problem better in a vast, complex, multimodal, and non-differentiable surface [76]. Additionally, it does not depend on gradient information of the error function and thus is particularly appealing when this information is difficult to obtain. These two advantages allowed the use of evolutionary-based methods to train different kind of ANNs, including recurrent ANNs [26, 42] and higher order ANNs [12].

Subsequently, a second strand of research investigated the design of ANN architectures, namely the number of neurons, the number of hidden layers, and the connectivity among the neurons. Differently with respect to the previous studies, where it was assumed that the architecture of an ANN is predefined and fixed during the evolution of connection weights, the main aim here is to use EC techniques for evolving the topology of the network. The problem can be formulated as an optimization problem in which the objective is to determine the global optimum into a search space where each point represents an architecture. Given an objective function to be optimized (i.e., a function that quantifies the quality or performance of each architecture), the design of the optimal topology corresponds in determining the highest point on the surface induced by the objective function on the search space of the architectures. This surface (or fitness landscape), presents some properties that make EC a suitable approach for finding the sought architecture [52]. The following features were identified and discussed by Miller et al. [52]: (1) the number of possible neurons and connections is unbounded, thus the fitness landscape is infinitely large; (2) since changes in the number of neurons or connections must be discrete, the surface is non-differentiable, thus making gradient-based approaches impossible to be used; (3) the mapping from network design to network performance after learning, is indirect, strongly epistatic, and dependent on initial conditions (e.g., initial weights), so the surface is complex and noisy; (4) structurally similar networks can show very different information processing capabilities, thus making the surface highly deceptive; and (5) structurally dissimilar networks can provide similar performance, thus the surface is multimodal.

For all these reasons, EC techniques seem to be a natural choice for addressing the problem at hand, and they represent an alternative approach, with respect to constructive and destructive algorithms, toward the automatic design of architectures. A constructive algorithm [15, 13] is a hill climbing method that starts with a minimal network (an ANN with a minimal number of hidden layers, nodes, and connections) and adds new layers, nodes, and connections during the training phase when deemed necessary and based on some criterion. On the other hand, destructive algorithms [56, 66, 8] search for the optimal topology starting with a maximal network and by subsequently removing unnecessary layers, nodes, and connections during the training phase [76]. While these approaches are simpler to implement with respect to EC-based methods, they are susceptible to becoming trapped at

structural local optima and they are able to explore only a small fraction of the possible ANN topologies [4].

Several works based on EC for optimizing the topology of an ANN appeared in the literature. One of the first works was proposed by Miller et al. [52], where a method based on a genetic algorithm is used for evolving neural network architectures for specific tasks. Each network architecture is represented as a connection constraint matrix mapped directly into a bit-string genotype. Modified standard genetic operators act on populations of these genotypes to produce network architectures with higher fitnesses over successive generations. Architecture fitness is assessed by training particular network instantiations and recording their final performance error. Schaffer et al. [63] demonstrated, using a genetic algorithm, that an evolved network architecture performs better than a large network using backpropagation learning alone when the criterion is correct generalization from a set of examples. In the same line of research, Wilson [75] showed that when genetic search is applied, a set of perceptrons can learn more complex tasks than initially apparent. Schiffmann et al. [64] presented a crossover operator for a genetic algorithm specifically created for automatic topology-optimization. In contrast to competing approaches, it allows that two parent networks with a different number of units can mate and produce a (valid) child network, which inherits genes from both of the parents. Similarly, Alba et al. [3] relied on a genetic algorithm to address the connectivity and structure definition problems, in order to accomplish a full genetic ANN design. Nikolopoulos and Fellrath [57] proposed the use of genetic algorithms and classifier systems to optimize the architecture of an ANN to be used for investment advising.

In the methods previously discussed, only the architecture of the ANN is evolved, but it is assumed that the activation function of each node in the architecture is fixed and predefined a priori. Despite the simplicity of this assumption, some studies demonstrated that the choice of the activation function plays a important role in determining the performance of an ANN [48, 9].

An important attempt to evolve the architecture of an ANN, as well as the activation functions, was proposed by Schoenauer and Ronald [65], where authors investigated the tuning of the slopes of the transfer functions of the individual neurons in the ANN. White and Ligomenides [72] adopted a simpler approach to the evolution of both topological structures and node transfer functions. The initial population contained ANNs with an 80% of the neurons using the sigmoid function and a 20% of the neurons using a Gaussian function. The evolutionary process was used to determine the optimal blend of these two functions in an automatic fashion. The idea of evolving the activation functions is nowadays investigated given the popularity of deep learning. For instance, the rectified linear activation (ReLU) function [33] has simplified the training of deep neural networks by counteracting the problems related to weight initialization and the vanishing gradient. As summarized by Manessi and Rozza [47], variations of ReLU have been proposed over the

years, such as leaky ReLU (LReLU) [46], which addresses dead neuron issues in ReLU networks, thresholded ReLU [38], which tackles the problem of large negative biases in autoencoders, and parametric ReLU (PReLU) [27], which treats the leakage parameter of LReLU as a per-filter learnable weight. While these works introduced new and useful activation functions, other works used more advanced strategies to learn the most suitable activation function for the particular architecture at hand. Agostinelli et al. [2] designed a novel form of piecewise linear activation function that is learned independently for each neuron using gradient descent. With this adaptive activation function, they were able to improve upon deep neural network architectures composed of static rectified linear units, achieving state-of-the-art performance on CIFAR-10, CIFAR-100, and a benchmark from high-energy physics involving Higgs boson decay modes. Manessi and Rozza [47] introduced two approaches to automatically learn different combinations of base activation functions (such as the identity function, ReLU, and hyperbolic tangent) during the training phase. They presented a thorough comparison of their novel approaches with well-known architectures on three standard datasets showing substantial improvements in the overall performance. Thus, the evolution of the activation functions is nowadays deemed as important as the evolution of the architectures of the ANNs [47].

The evolutionary methods just discussed only evolve the architecture of ANNs, without any connection weights. That is, connection weights have to be learned in a subsequent step. While this approach reduces the complexity of evolving both the topology and the weights, there is a major problem with the evolution of architectures without connection weights as pointed out by Yao and Liu [77]. In particular it is possible to identify two critical issues: (1) different random initial weights may produce different training results. Thus, the same genotype may have different fitness due to different random initial weights used in training; and (2) different training algorithms may produce different training results even from the same set of initial weights. This is especially true for multimodal error functions.

Thus, the remaining part of this section recalls contributions where EC-based techniques were used to optimize the weights and the topology of an ANNs simultaneously. The idea behind this approach is that each individual in a population is a fully specified ANN with complete weight information. As a consequence, there is a one-to-one mapping between a genotype and its phenotype, thus allowing the search process to overcome the issues related to the fitness evaluation. Srinivas and Patnaik [68] presented a technique for reducing the search space of the genetic algorithm to improve its performance in searching for the globally optimal set of connection weights. They used the notion of equivalent solutions in the search space, and included in the reduced search-space only one solution, called the base solution, from each set of equivalent solutions. The iteration of the genetic algorithm consisted of an additional step where the solutions are mapped to the respective base solutions. A genetic algorithm based method was also proposed by Bornholdt and

Grauden [5] for evolving a network that represented a model for a brain with sensory and motor neurons. Oliker et al. [58] proposed a distributed genetic algorithm for designing and training neural networks. The method sets the neural network's architecture and weights for a given task where the network is comprised of binary linear threshold units. White and Ligomenides [72] introduced a new algorithm which uses a genetic algorithm to determine the topology and link weights of a neural network. If the genetic algorithm fails to find a satisfactory solution network, the best network developed by the genetic algorithm is used to try to find a solution via back-propagation. In this way, each algorithm is used to its greatest advantage: the genetic algorithm (with its global search) determines a (sub-optimal) topology and weights to solve the problem, and back-propagation (with its local search) seeks the best solution in the neighborhood of the weight and topology spaces found by the genetic algorithm.

Besides genetic algorithms, other EC methods were used to address the optimization problem at hand. Koza and Rice [39] showed how to use genetic programming to find both the weights and architecture for a neural network, including the number of layers, the number of processing elements per layer, and the connectivity between processing elements. Jian and Yugeng [34] presented a new design method for the structure and weights of static neural networks based on evolutionary programming. The method is further extended to design recurrent neural networks through introducing delayed links into networks. Particle Swarm Optimization (PSO) has also been used to evolve both the weights and the topology of the networks [78, 37, 17]. In particular, Kiranyaz et al. [37] presented a Multi-Dimensional Particle Swarm Optimization (MD-PSO) technique for the automatic design of Artificial Neural Networks by evolving to the optimal network configuration (connections, weights, and biases) within the architecture space. Similarly, Garro and Vázquez [17] explored the simultaneous evolution of the three principal components of an ANN: the set of synaptic weights, the connections or architecture, and the transfer function for each neuron. The main topic of this contribution was the evaluation of eight different proposed fitness functions used to evaluate the quality of each solution and find the best design. Zhang et al. [78] introduced a new evolutionary system for evolving Feed-Forward ANNs, which is constrained to the use of PSO. Both the architecture and the weights of ANNs were adaptively adjusted according to the quality of the network. One of the most popular and broadly used approaches in neuroevolution is the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [70]. Recently, NEAT has been evolved to CoDeepNEAT [51], an algorithm capable of covering more complex areas such as vision, speech and language.

To conclude, among the many existing references reporting on the use of EC to optimize ANNs, the reader is particularly referred to [76, 61, 69] for a comprehensive overview of this research field.

3 Semantic Learning Machine

3.1 Algorithm

In the proposal of Geometric Semantic Genetic Programming (GSGP), Moraglio et al. [54] showed that any supervised learning problem where the error is measured as a distance to the known targets has a unimodal error landscape. This property can be exploited by constructing specific variation operators. These operators are known as geometric semantic operators. In this context, the term semantics is used to refer to the outputs of any supervised learning model (e.g., a neural network) over a set of data instances. In GSGP, geometric semantic operators were defined for some domains: boolean, arithmetic, and conditional rules. GSGP was shown to outperform the traditional syntactic genetic programming approach in several datasets [54, 19].

The reasoning behind these geometric semantic operators can be used to create equivalent operators for other representations or computational models. With the proposal of the Semantic Learning Machine (SLM) neuroevolution algorithm [20, 23], it is achievable to perform semantic search for the space of Neural Networks (NNs). This was made possible by deriving the arithmetic mutation from GSGP to the space of NNs, therefore defining a geometric semantic mutation for NNs. This allows SLM to effectively and efficiently explore the space of NNs by exploiting the underlying unimodal error landscape. Given that these error landscapes are unimodal, no local optima exist. In the case of SLM this means that, with the exception of the global optimum, every point in the search space has at least one neighbor with better fitness, and that neighbor is reachable through the application of the mutation operator. The direct consequence is that a hill climbing strategy can effectively advance the search.

SLM is essentially a geometric semantic hill climber for NNs that follows a $(1+\lambda)$ strategy. Without local optima, the search can be focused around the current best NN without incurring in any particular disadvantage. SLM can be summarized in the following steps:

1. Generate N initial random NNs
2. Choose the best NN (B) from the initial random NNs, according to the selected performance criterion
3. Repeat the following steps until a given stopping criterion is met:
   a. Apply the geometric semantic mutation to the current best (B) N times to generate N new NNs (known as children or neighbors)
   b. Update B as being the NN with the best performance according to the selected criterion, considering the current B and the N newly generated NNs
4. Return B as the best performing NN according to the selected performance criterion

The initial random NNs can be generated without any particular restriction. They can have any number of layers and neurons, with any activation functions, while the weights in the connections between the neurons can be freely selected. The networks do not have to be fully-connected and can be as sparsely connected as desired. The crucial aspect of SLM is the geometric semantic mutation which takes a parent NN and produces a child NN. This mutation works by adding new hidden neurons while ensuring that the semantics of the parent's hidden neurons are not affected by these new hidden neurons. To ensure this fundamental aspect of the geometric semantic mutation, the new hidden neurons do not feed their computations to the parent's hidden neurons, with the exception of the output neurons. The weights of connections from the new hidden neurons in the last hidden layer to the output neurons are defined by the learning step. The learning step can be computed optimally with the Moore-Penrose pseudoinverse (similarly to the case of GSGP [19, 55, 21]), or it can be defined as a parameter to be tuned. Each new hidden neuron added can select from which neurons it receives incoming connections. This means that the sparseness level can be easily controlled by defining how many incoming connections each new neuron will receive. The weights of each connection can be freely selected as in the initialization step. As is common in neuroevolution algorithms, SLM does not rely on backpropagation to adjust the weights of the NNs. For further SLM details the reader is referred to Gonçalves et al. [20] and Gonçalves [23].

3.2 Previous Comparisons with Other Neuroevolution Methods

Jagusch et al. [32] explored several SLM variants and performed a comparison with other neuroevolution methods as well as other well-established supervised machine learning techniques. Regarding the neuroevolution methods, NEAT and a fixed-topology neuroevolution approach were used as points of comparison. NEAT was one the focus of the comparison given its popularity. The comparisons were performed on a total of nine real-world datasets freely available from the UCI Machine Learning Repository [43]: four binary classification datasets and five regression datasets. The results showed that, in terms of learning the training data, SLM was superior to the other neuroevolution methods in all the nine datasets considered (all with statistically significant differences). In this comparison the best SLM variant was, naturally, always the one that computed the optimal learning step. Focusing on the NEAT comparison and on the generalization performance, SLM was found to be superior to NEAT, with statistically significant differences, in eight out of the nine datasets considered. No statistically significant difference was found in the remaining dataset. Furthermore, particularly the SLM variant that computed the optimal learning step and used a semantic stopping criterion [25] (further details on section 4.2) also resulted in much smaller neural networks

and achieved speed-ups of various orders of magnitude over NEAT on several datasets.

4 Experimental Methodology

4.1 Datasets and Parameter Tuning

In the experimental phase, four real-world binary classifications datasets are considered: Cancer, Credit, Diabetes, and Sonar. In Credit, the objective is to classify the individuals as either good or bad credit whereas in Diabetes and Cancer, the goal is to predict whether an individual has diabetes or cancer, respectively. The Sonar task aims at classifying sonar signals as they either bounced off a metal cylinder or a roughly cylindrical rock. All of these datasets are freely available from the UCI Machine Learning Repository [43]. Table 1 presents the number of features (input variables), the number of instances (observations), and the % of class 1 instances in each of the four datasets under consideration.

Table 1 Binary classification datasets considered

Dataset	Features	Instances	% of class 1 instances
Cancer	30	569	$\approx 37\%$
Credit	24	1000	30%
Diabetes	8	768	$\approx 35\%$
Sonar	60	208	$\approx 47\%$

Different SLM variants are compared with the Multi-layer Perceptron (MLP) trained with backpropagation. A nested k-fold cross-validation (CV) methodology is followed. A 30-fold outer CV is used to obtain 30 final generalization values (test set values) to assess the statistical significance of the results. For each outer training fold, a 2-fold inner CV is conducted to perform parameter tuning for each algorithm. Both algorithms are allowed to explore a total of 72 random parameter combinations during parameter tuning. Four SLM and two MLP variants are tested (detailed on sections 4.2 and 4.3). To ensure fairness, SLM tests 18 parameter combinations for each of the four variants considered, while MLP tests 36 parameter combinations for each of the two variants considered.

4.2 SLM Variants

The base SLM configuration is the following:

- In the initial population each NN is generated with a random number of hidden layers selected between 1 and 5
- In the initial population each NN randomly selects the number of neurons for each hidden layer between 1 and 5
- Each hidden neuron randomly selects its activation function from the following options: Logistic, Relu, and Tanh
- Each hidden neuron randomly selects the weight of each incoming connection from values in the interval $[-mncw, mncw]$, where $mncw$ represents the maximum neuron connection weight parameter (subject to parameter tuning)
- Each hidden neuron randomly selects the weight of its bias from values in the interval $[-mbw, mbw]$, where mbw represents the maximum bias weight parameter (subject to parameter tuning)
- Each time a new NN is created by the mutation operator, the number of new neurons to be added to each layer is randomly selected between 1 and 3

The three main differences between the SLM variants under study are the following: (1) the strategy regarding the learning step; (2) the type of training data use (static or dynamic); (3) the stopping criterion to decide the termination of the training process.

Regarding the learning step, two variants are considered: computing the Optimal Learning Step (OLS) for each application of the mutation operator; and using a Bounded Learning Step (BLS). The SLM-BLS variants introduce an additional parameter that defines the maximum learning step (mls) that bounds the learning step. At each application of the mutation operator, the effective learning step is randomly selected from values in the interval $[-mls, mls]$.

In terms of dynamic training data use, two approaches are considered: randomly selecting a subset of the training data at each iteration and computing the quality of each solution with this subset; and always using the complete training data but randomly weighting each instance (between 0 and 1) and changing these weights at each iteration. The first approach is referred to as Random Sampling Technique (RST) following Gonçalves et al. [22, 18, 24], while the second approach is referred to as Random Weighting Technique (RWT). In genetic programming, RST successfully contributed to avoid overfitting and improve generalization on high-dimensional datasets [22, 18]. Other studies using dynamic training data in genetic programming have followed [50, 49, 16, 67].

Finally, regarding the termination of the training process, the following approaches are considered: termination based on a given number of iterations; and termination based on a semantic stopping criterion. The Semantic Stopping Criteria (SSC) proposed by Gonçalves et al. [25] use information gathered from the semantic neighborhood (the set of new models generated by the mutation) to decide when to stop the search. These are named the

Error Deviation Variation (EDV) criterion and the Training Improvement Effectiveness (TIE) criterion. EDV stops the search when a considerable majority of the neighbors are improving the training performance at the expense of larger error deviations. TIE stops the search when training error improvements become harder to find within the semantic neighborhood. This can signal that the training error improvements are being forced at the expense of the resulting generalization. These SSC can be used to avoid setting a maximum number of iterations and to avoid setting data aside to use as a validation set to decide when to stop.

With these different aspects into consideration, the following SLM variants are grouped and named as follows:

1. BLS variants: SLM-BLS, SLM-BLS + RST, and SLM-BLS + RWT
2. OLS variants: SLM-OLS, SLM-OLS + RST, and SLM-OLS + RWT
3. BLS + TIE/EDV: SLM-BLS + TIE/EDV
4. OLS + EDV: SLM-OLS + EDV

When SLM-BLS is mentioned by itself it refers to SLM-BLS without using RST and RWT. Similarly, when SLM-OLS is mentioned by itself it refers to SLM-OLS without using RST and RWT.

All SLM variants can tune the maximum neuron connection weight (*mncw*) and the maximum bias weight (*mbw*) in the range $[0.1, 0.5]$. The BLS variants and BLS + TIE/EDV can tune the maximum learning step (*mls*) in the range $[0.1, 2]$, and the number of iterations in the range $[1, 100]$. The BLS and the OLS variants select with equal probability the use of RST, RWT, or none. BLS + TIE/EDV selects with equal probability the use of EDV or TIE as the semantic stopping criterion. Whenever RST is used, the parameter that defines the ratio of the total training data to be used (the subset ratio) is selected from the range $[0.01, 0.99]$.

4.3 MLP Variants

Two MLP variants are considered: the most common stochastic gradient descent (SGD) [59, 35] variant, and the Adam SGD variant [36]. For SGD and Adam, the following parameters are tuned:

- The number of iterations in the range $[1, 100]$
- The batch size between 50 and the maximum number of training instances available
- The activation function to be used in the hidden layers: Logistic, Relu, and Tanh
- The number of hidden layers in the range $[1, 5]$
- The number of hidden neurons per layer in the range $[1, 200]$
- The learning rate in the range $[0.1, 2]$

- The L2 penalty in the range $[0.1, 10]$

SGD can also select the momentum in the range $[0.0000001, 1]$ and decide to use or not the Nesterov's momentum. Adam can also select the beta 1 and beta 2 parameters in the range $[0, 1[$.

5 Results and Analysis

This section analyzes the results obtained in the experimental phase. Section 5.1 presents the results achieved by the different variants of the SLM algorithm taken into account, analyzing the performance obtained on the validation set and discussing some aspects related to the choice of the parameters. Subsequently, section 5.2 presents the results produced by the MLP over the same benchmark problems and discusses the main performance differences between MLP and SLM. Section 5.3 compares SLM and MLP after their best configuration are found and explores the generalization ability of SLM under different ensemble construction methods.

5.1 SLM

This section presents the results obtained by considering different groups of SLM variants. The first analysis refers to the validation Area Under Receiver Operating Characteristic (AUROC) curve values produced by the considered SLM variants, and the results are summarized in Table 2. For each benchmark problem and for each technique, this Table reports the mean and the standard deviation of the validation AUROC. These values were obtained from the nested cross-validation procedure previously described. Thus they are the mean and standard deviation values achieved by the best models obtained in the inner cross-validation procedure (that was performed to determine the most suitable values of the hyperparameters). According to the results reported in this Table and complementing them with the ones in Table 3, it is possible to state that OLS variants are the best performer, outperforming the other variants taken into account. In particular, the OLS variants outperformed the other competitors 23 times on both the Cancer and the Credit datasets, 20 times on the Diabetes dataset, and 26 times on the Sonar dataset. The second-best performer when considering the Cancer and the Sonar datasets is the BLS variants group, while OLS + EDV outperforms the other competitors 7 times on the Credit dataset and 9 times on the Diabetes dataset. BLS + TIE/EDV performs poorly in relative terms. A possible explanation might be that, for the datasets considered, the maximum number of iterations is not high enough for the semantic stopping criterion to take effect under a bounded learning step. Overall, OLS families (OLS variants and

OLS + EDV) seem to provide higher AUROC values with respect to the BLS families. A global view on the average AUROC values reported in Table 2 suggests that: (1) Within the BLS groups, BLS + TIE/EDV always achieved a lower average validation AUROC than the BLS variants; (2) Within the OLS groups, OLS + EDV is the best performer on the Cancer dataset, while the OLS variants group is the best performer over the remaining benchmarks taken into account; (3) the OLS variants represent the most suitable choice for the classification problems at hand.

The subsequent analysis considers the average number of iterations performed by each SLM variant. Results of this analysis are reported in Table 4. The BLS variants require a larger number of iterations with respect to the other competitors. This behavior was expected considering that the BLS variants do not use any semantic stopping criterion and optimal learning step. Focusing on the other competitors, the OLS variants (the best performer over these problems) ran for a significantly larger number of iterations with respect to BLS + TIE/EDV. When comparing OLS variants with OLS + EDV, it is possible to see that also in this case the number of iterations performed by OLS + EDV is significantly lower than the one of the OLS variants. The use of the optimal learning step allows OLS + EDV to reach satisfactory performance in all of the considered benchmarks and to outperform the OLS variants over the Cancer dataset. To summarize the results of this analysis, it seems that the use of a semantic stopping criterion with the OLS and the BLS is effective at reducing the computational effort needed to perform the training process, but according to the problem at hand, it may not result in the best overall performance.

With respect to the semantic stopping criterion, Table 5 compares the usage of EDV and TIE in SLM-BLS. According to these values, it is clear that EDV is more effective than the TIE stopping criterion in the classification problems considered. An additional analysis performed during the experimental phase aimed at understanding whether the random weighting/sampling techniques are beneficial when coupled with the SLM variants. Results of this analysis are reported in Table 6, where the use of RST, RWT, and the whole original set of observations (None) are compared in the context of BLS variants and OLS variants. According to these values, it seems that RWT is more effective than RST in both of the considered SLM variants. Comparing the use of RST and RWT with the use of the whole original set of observations, the results of Table 6 suggest that random sampling and random weighting techniques can improve the performance of SLM. This shows that the dynamic use of training data can indeed be beneficial within SLM. This is particularly clear when RWT is used in conjunction with the optimal learning step computation.

Table 2 Validation AUROC for each SLM variant considered

Dataset	BLS variants	OLS variants	BLS + TIE/EDV	OLS + EDV
Cancer	0.951 +- 0.095	0.937 +- 0.124	0.896 +- 0.185	0.959 +- 0.061
Credit	0.679 +- 0.134	0.733 +- 0.120	0.564 +- 0.166	0.688 +- 0.108
Diabetes	0.680 +- 0.169	0.784 +- 0.110	0.626 +- 0.153	0.738 +- 0.131
Sonar	0.648 +- 0.282	0.816 +- 0.213	0.636 +- 0.277	0.724 +- 0.220

Table 3 Best SLM configuration by variant

Dataset	BLS variants	OLS variants	BLS + TIE/EDV	OLS + EDV
Cancer	5	23	0	2
Credit	0	23	0	7
Diabetes	1	20	0	9
Sonar	2	26	1	1

Table 4 Number of iterations for each SLM variant considered

Dataset	BLS variants	OLS variants	BLS + TIE/EDV	OLS + EDV
Cancer	79.567 +- 16.332	64.067 +- 23.712	3.067 +- 2.741	1.133 +- 0.434
Credit	68.267 +- 20.793	63.433 +- 26.165	3.967 +- 2.399	2.233 +- 1.406
Diabetes	76.000 +- 18.819	60.467 +- 22.508	5.567 +- 8.336	1.967 +- 1.217
Sonar	75.033 +- 22.172	58.667 +- 24.288	3.533 +- 3.391	3.167 +- 4.900

Table 5 EDV and TIE use in SLM-BLS

Dataset	EDV	TIE
Cancer	27	3
Credit	26	4
Diabetes	25	5
Sonar	25	5

5.2 MLP

This section presents the results obtained by both MLP variants: Adam and SGD. The first part of this discussion considers the performance on the validation set and the corresponding results are reported in Table 7. According to these values, Adam is the best performer over the Cancer dataset, while SGD is the best performer over the Credit dataset and the Sonar dataset. The two MLP variants produce the same performance on the Diabetes dataset. According to these results it is difficult to draw a general conclusion, and it

Table 6 RST and RWT use in the BLS and the OLS variants

Dataset	BLS variants			OLS variants		
	None	RST	RWT	None	RST	RWT
Cancer	8	9	13	15	6	9
Credit	18	4	8	15	4	11
Diabetes	11	6	13	12	4	14
Sonar	15	4	11	15	6	9

seems that the choice between Adam and SGD must be evaluated according to the particular problem at hand. To complement this analysis, Table 8 shows the best MLP configurations by variant. According to these values, SGD produces the best performance most of the times over the Credit and Sonar dataset, while Adam returns the best performance on the remaining datasets in the vast majority of the runs considered.

At this stage, it is important to compare the results of Table 7 (obtained with MLP) with the ones reported in Table 2 (obtained with SLM). According to these results, all the SLM variants are able to outperform the best MLP variant over all the classification problems under exam. This comparison clearly demonstrates the superiority of SLM (with respect to MLP) in creating models characterized by a greater validation AUROC. Overall, the SLM algorithm is a competitive option to consider in these classification problems given that its performance (independently of the selected variant) is significantly better than the best MLP variant.

Another important aspect to analyze in the comparison between MLP and SLM, is the number of iterations required to produce the final model. While this analysis was performed for the SLM algorithm (see Table 4), Table 9 reports the same information for MLP-based models. In particular, the SLM variants that use a semantic stopping criterion are able to build a classification model in a considerably lower amount of iterations with respect to an MLP variant based on the backpropagation algorithm. This smaller number of iterations does not negatively affect the performance of the final models, as these SLM variants are able to outperform MLP over all the considered benchmarks.

To further understand the different MLP variants, Table 10 reports the activation function used by each one of them. From these values it is interesting to point out that the Adam variant has a clear preference for the Relu function, disregarding the problem under analysis. SGD shows a preference for the Relu function when considering the Cancer and Diabetes datasets, while Tanh is the function used most of the times over the Credit dataset. For the Sonar dataset the Logistic function was never selected and Relu and Tanh were selected 14 and 16 times respectively.

Table 7 Validation AUROC for each MLP variant considered

Dataset	Adam	SGD
Cancer	0.542 +- 0.110	0.500 +- 0.000
Credit	0.509 +- 0.031	0.525 +- 0.048
Diabetes	0.498 +- 0.016	0.498 +- 0.011
Sonar	0.496 +- 0.023	0.581 +- 0.109

Table 8 Best MLP configuration by variant

Dataset	Adam	SGD
Cancer	28	2
Credit	9	21
Diabetes	24	6
Sonar	7	23

Table 9 Number of iterations for each MLP variant considered

Dataset	Adam	SGD
Cancer	56.200 +- 25.183	55.500 +- 26.046
Credit	56.400 +- 24.210	57.167 +- 26.592
Diabetes	42.433 +- 27.320	56.700 +- 30.326
Sonar	50.267 +- 24.237	49.667 +- 29.352

Table 10 Activation functions use by MLP variant

Dataset	Adam			SGD		
	Logistic	Relu	Tanh	Logistic	Relu	Tanh
Cancer	8	15	7	7	16	7
Credit	9	13	8	9	8	13
Diabetes	4	16	10	7	15	8
Sonar	8	17	5	0	14	16

5.3 Generalization and Ensemble Analysis

This section starts by assessing the generalization (i.e., the test set performance over the 30 outer folds) of SLM and MLP considering the best parameter configurations found after parameter tuning. Figure 1 presents the boxplots for the AUROC values of both algorithms. On each box, the central mark is the median, the edges of the box are the 25^{th} and 75^{th} percentiles, and the whiskers extend to the most extreme data points that are not considered outliers.

These boxplots show that SLM consistently achieves better AUROC values than MLP across all datasets. A set of statistical tests is performed to assess the statistical significance of these results. Firstly, a Kolmogorov-Smirnov test is applied to assess if these values come from a normal distribution. The result of this test suggests that the alternative hypothesis (i.e., the data do not come from a normal distribution) cannot be rejected considering a significance level (α) of 0.05. Given this outcome, a rank-based statistic is selected for the next step. A Mann-Whitney U-test is performed with the null hypothesis that the samples have equal medians. As in the previous test, a significance level of 0.05 is considered. The outcomes suggest that SLM outperforms MLP in all datasets with statistically significant differences. The p-values for these comparisons can be found in table 11.

The final part of the analysis studies the outcomes of different ensemble construction methods when using SLM as a base learner. Bagging [6] and

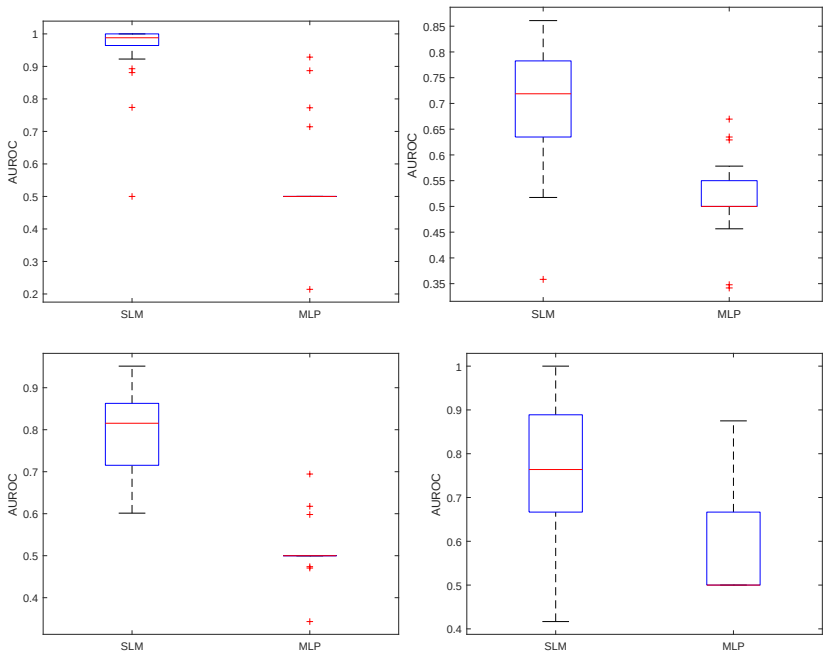

Fig. 1 Boxplots for test set AUROC values of SLM and MLP: Cancer, Credit, Diabetes, and Sonar

Table 11 p-values of Mann-Whitney U-tests over test set AUROC values of SLM and MLP

Dataset	p-value
Cancer	3.486×10^{-11}
Credit	1.054×10^{-8}
Diabetes	1.036×10^{-11}
Sonar	9.894×10^{-5}

Boosting [11] methods are compared with a common simple averaging construction method that trains the base learner N times without changing the training instances provided. This simpler ensemble construction method can be effective if the base learner already has an inherent diversity within its search process. This might be the case of SLM given that it is a stochastic algorithm. These three ensemble construction methods are used to create ensembles of 30 NNs using SLM as the base learner. In the Boosting case, four variations of AdaBoost.R2 are studied and labeled as follows:

- Boosting-1: weighted median prediction and fixed learning rate of 1
- Boosting-2: weighted median prediction and variable learning rate selected randomly in the interval $[0, 1]$ for each new NN added to the ensemble
- Boosting-3: weighted mean prediction and fixed learning rate of 1

- Boosting-4: weighted mean prediction and variable learning rate selected randomly in the interval $[0, 1]$ for each new NN added to the ensemble

Figure 2 presents the boxplots for the test set AUROC values of each ensemble construction method considered: Simple (averaging), Bagging, and the four Boosting variations.

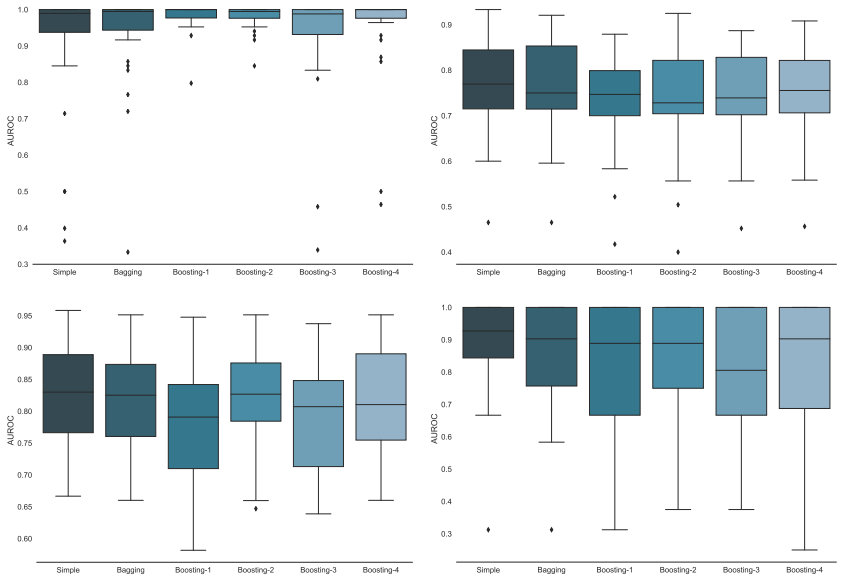

Fig. 2 Boxplots for test set AUROC values of each ensemble construction method considered: Cancer, Credit, Diabetes, and Sonar

These ensemble results show that the simple averaging method performs similarly to Bagging and Boosting. In terms of the median AUROC value, the simple averaging method even achieves the highest value in three of the four datasets: Credit, Diabetes, and Sonar. Overall, these different ensemble construction methods perform similarly in terms of the distribution of the values. These results suggest that the stochastic nature of SLM allows the simple averaging method to perform well without having to explicitly confer more diversity to the base learner, e.g., by providing different training instances to each ensemble member.

6 Toward the Deep Semantic Learning Machine

Recently, SLM was used in conjunction with Convolutional Neural Networks (CNNs) for the first time [41, 40]. In these contributions, the task of discrim-

inating between benign and malignant prostate cancer lesions given multiparametric magnetic resonance imaging was addressed. This image classification task was proposed in the context of the PROSTATEx competition [44]. SLM was tested as a backpropagation replacement for the training of the last fully-connected layers of CNNs. In this approach, the outputs from the convolutional layers of a given CNN are passed as inputs (without pre-training) to SLM. The empirical comparison is performed with XmasNet [45], a state-of-the-art CNN specifically developed to address the PROSTATEx 2017 competition. The results show that SLM achieves higher AUROC curve values than XmasNet with a statistically significant difference. This performance is achieved without neither pre-training the underlying CNN nor relying on backpropagation. Furthermore, SLM is also much more computationally efficient in the training phase. SLM achieves an average speed-up of around 14 over training with backpropagation. This is also important as neuroevolution methods are sometimes perceived as slow. Additionally, it is important to emphasize that SLM was only run on CPU (whereas XmasNet was trained using a GPU) and without any explicit parallelization. This further reinforces the results obtained, given that each network evaluation could be suitably parallelized, thus achieving a higher speed-up. Furthermore, inside each network evaluation, the new nodes of a given layer can also be evaluated in parallel. SLM can be further extended to include the convolutional layers within the search process. This would remove the need for a fixed CNN topology to be provided and it would also eliminate the burden of assessing several CNN topologies in order to find a suitable topology for the task at hand. Adapting SLM to include the convolutional layers within the search would result in a unimodal search over the space of CNNs. Such a development could result in considerable improvements in the field of deep learning and computer vision. This is something that is currently under study.

Acknowledgements This work was partially supported by projects UID/MULTI/00308/2019 and by the European Regional Development Fund through the COMPETE 2020 Programme, FCT - Portuguese Foundation for Science and Technology and Regional Operational Program of the Center Region (CENTRO2020) within project MAnAGER (POCI-01-0145-FEDER-028040). This work was also partially supported by national funds through FCT (Fundação para a Ciência e a Tecnologia) under project DSAIPA/DS/0022/2018 (GADgET).

References

1. Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A., Arshad, H.: State-of-the-art in artificial neural network applications: A survey. Heliyon 4(11), e00,938 (2018)
2. Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P.: Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830 (2014)

3. Alba, E., Aldana, J., Troya, J.M.: Full automatic ann design: A genetic approach. In: International Workshop on Artificial Neural Networks, pp. 399–404. Springer (1993)
4. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks 5(1), 54–65 (1994)
5. Bornholdt, S., Graudenz, D.: General asymmetric neural networks and structure design by genetic algorithms. Neural networks 5(2), 327–334 (1992)
6. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996).
7. Chauvin, Y., Rumelhart, D.E.: Backpropagation: theory, architectures, and applications. Psychology Press (2013)
8. Cun, Y.L., Denker, J.S., Solla, S.A.: Advances in neural information processing systems 2. chap. Optimal Brain Damage, pp. 598–605. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1990)
9. DasGupta, B., Schnitger, G.: Efficient approximation with neural networks: A comparison of gate functions. Pennsylvania State University, Department of Computer Science (1992)
10. Dill, F.A., Deer, B.C.: An exploration of genetic algorithms for the selection of connection weights in dynamical neural networks. In: Proceedings of the IEEE 1991 National Aerospace and Electronics Conference NAECON 1991, vol. 3, pp. 1111–1115 (1991)
11. Drucker, H.: Improving regressors using boosting techniques. In: Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, pp. 107–115. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997).
12. Epitropakis, M.G., Plagianakos, V.P., Vrahatis, M.N.: Evolutionary Algorithm Training of Higher-Order Neural Networks. IGI Global (2009)
13. Fahlman, S.E., Lebiere, C.: Advances in neural information processing systems 2. chap. The Cascade-correlation Learning Architecture, pp. 524–532. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1990)
14. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evolutionary Intelligence 1(1), 47–62 (2008)
15. Frean, M.: The upstart algorithm: A method for constructing and training feedforward neural networks. Neural Computation 2(2), 198–209 (1990)
16. Galván-López, E., Vázquez-Mendoza, L., Schoenauer, M., Trujillo, L.: On the use of dynamic gp fitness cases in static and dynamic optimisation problems. In: International Conference on Artificial Evolution (Evolution Artificielle), pp. 72–87. Springer (2017)
17. Garro, B.A., Vázquez, R.A.: Designing artificial neural networks using particle swarm optimization algorithms. Computational intelligence and neuroscience 2015, 61 (2015)
18. Gonçalves, I., Silva, S.: Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In: Genetic Programming, pp. 73–84. Springer (2013)
19. Gonçalves, I., Silva, S., Fonseca, C.M.: On the generalization ability of geometric semantic genetic programming. In: Genetic Programming, pp. 41–52. Springer (2015)
20. Gonçalves, I., Silva, S., Fonseca, C.M.: Semantic learning machine: A feedforward neural network construction algorithm inspired by geometric semantic genetic programming. In: Progress in Artificial Intelligence, Lecture Notes in Computer Science, vol. 9273, pp. 280–285. Springer (2015)
21. Gonçalves, I., Silva, S., Fonseca, C.M., Castelli, M.: Arbitrarily close alignments in the error space: A geometric semantic genetic programming approach. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 99–100. ACM (2016)

22. Gonçalves, I., Silva, S., Melo, J.B., Carreiras, J.M.B.: Random sampling technique for overfitting control in genetic programming. In: Genetic Programming, pp. 218–229. Springer (2012)
23. Gonçalves, I.: An exploration of generalization and overfitting in genetic programming: Standard and geometric semantic approaches. Ph.D. thesis, Department of Informatics Engineering, University of Coimbra, Portugal (2017)
24. Gonçalves, I., Silva, S.: Experiments on controlling overfitting in genetic programming. In: Local proceedings of the 15th Portuguese Conference on Artificial Intelligence, EPIA 2011 (2011)
25. Gonçalves, I., Silva, S., Fonseca, C.M., Castelli, M.: Unsure when to stop? ask your semantic neighbors. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17, pp. 929–936. ACM, New York, NY, USA (2017).
26. Greenwood, G.W.: Training partially recurrent neural networks using evolutionary strategies. IEEE Transactions on Speech and Audio Processing 5(2), 192–194 (1997)
27. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034 (2015)
28. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural networks 4(2), 251–257 (1991)
29. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks 2(5), 359–366 (1989)
30. Hush, D.R., Horne, B.G.: Progress in supervised neural networks. IEEE signal processing magazine 10(1), 8–39 (1993)
31. Irie, B., Miyake, S.: Capabilities of three-layered perceptrons. In: IEEE International Conference on Neural Networks, vol. 1, p. 218 (1988)
32. Jagusch, J.B., Gonçalves, I., Castelli, M.: Neuroevolution under unimodal error landscapes: an exploration of the semantic learning machine algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 159–160. ACM (2018)
33. Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al.: What is the best multi-stage architecture for object recognition? In: 2009 IEEE 12th international conference on computer vision, pp. 2146–2153. IEEE (2009)
34. Jian, F., Yugeng, X.: Neural network design based on evolutionary programming. Artificial Intelligence in engineering 11(2), 155–161 (1997)
35. Kiefer, J., Wolfowitz, J.: Stochastic estimation of the maximum of a regression function. Ann. Math. Statist. 23(3), 462–466 (1952).
36. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2014).
37. Kiranyaz, S., Ince, T., Yildirim, A., Gabbouj, M.: Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. Neural networks 22(10), 1448–1462 (2009)
38. Konda, K., Memisevic, R., Krueger, D.: Zero-bias autoencoders and the benefits of co-adapting features. arXiv preprint arXiv:1402.3337 (2014)
39. Koza, J.R., Rice, J.P.: Genetic generation of both the weights and architecture for a neural network. In: IJCNN-91-seattle international joint conference on neural networks, vol. 2, pp. 397–404. IEEE (1991)
40. Lapa, P., Gonçalves, I., Rundo, L., Castelli, M.: Enhancing classification performance of convolutional neural networks for prostate cancer detection on magnetic resonance images: a study with the semantic learning machine. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '19. ACM, New York, NY, USA (2019).
41. Lapa, P., Gonçalves, I., Rundo, L., Castelli, M.: Semantic learning machine improves the cnn-based detection of prostate cancer in non-contrast-enhanced mri. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '19. ACM, New York, NY, USA (2019).

42. Lei, J., He, G., Jiang, J.P.: The state estimation of the cstr system based on a recurrent neural network trained by hgas. In: Proceedings of International Conference on Neural Networks (ICNN'97), vol. 2, pp. 779–782 (1997)
43. Lichman, M.: UCI machine learning repository (2013).
44. Litjens, G., Debats, O., Barentsz, J., Karssemeijer, N., Huisman, H.: "PROSTATEx Challenge data", The Cancer Imaging Archive. https://wiki.cancerimagingarchive.net/display/Public/SPIE-AAPM-NCI+PROSTATEx+Challenges (2017). Online; Accessed on January 25, 2019
45. Liu, S., Zheng, H., Feng, Y., Li, W.: Prostate cancer diagnosis using deep learning with 3D multiparametric MRI. In: Medical Imaging 2017: Computer-Aided Diagnosis, Proceedings SPIE, vol. 10134, p. 1013428. International Society for Optics and Photonics (2017).
46. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml, vol. 30, p. 3 (2013)
47. Manessi, F., Rozza, A.: Learning combinations of activation functions. In: 2018 24th International Conference on Pattern Recognition (ICPR), pp. 61–66. IEEE (2018)
48. Mani, G.: Learning by gradient descent in function space. In: 1990 IEEE International Conference on Systems, Man, and Cybernetics Conference Proceedings, pp. 242–247 (1990)
49. Martínez, Y., Naredo, E., Trujillo, L., Legrand, P., López, U.: A comparison of fitness-case sampling methods for genetic programming. Journal of Experimental & Theoretical Artificial Intelligence 29(6), 1203–1224 (2017)
50. Martinez, Y., Trujillo, L., Naredo, E., Legrand, P.: A comparison of fitness-case sampling methods for symbolic regression with genetic programming. In: EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V, pp. 201–212. Springer (2014)
51. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., Hodjat, B.: Evolving deep neural networks. In: R. Kozma, C. Alippi, Y. Choe, F.C. Morabito (eds.) Artificial Intelligence in the Age of Neural Networks and Brain Computing. Amsterdam: Elsevier (2018).
52. Miller, G.F., Todd, P.M., Hegde, S.U.: Designing neural networks using genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, pp. 379–384. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
53. Montana, D.J., Davis, L.: Training feedforward neural networks using genetic algorithms. In: IJCAI, vol. 89, pp. 762–767 (1989)
54. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Parallel Problem Solving from Nature-PPSN XII, pp. 21–31. Springer (2012)
55. Moraglio, A., Mambrini, A.: Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation, pp. 989–996. ACM (2013)
56. Mozer, M.C., Smolensky, P.: Advances in neural information processing systems 1. chap. Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment, pp. 107–115. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
57. Nikolopoulos, C., Fellrath, P.: A hybrid expert system for investment advising. Expert Systems 11(4), 245–250 (1994)
58. Oliker, S., Furst, M., Maimon, O.: Design architectures and training of neural networks with a distributed genetic algorithm. In: IEEE International Conference on Neural Networks, pp. 199–202. IEEE (1993)

59. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Statist. 22(3), 400–407 (1951).
60. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature 323(6088), 533 (1986)
61. S. Ding H. Li, C.S.J.Y.F.J.: Evolutionary artificial neural networks: a review. Artificial Intelligence Review 39, 251–260 (2013).
62. Samarasinghe, S.: Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition. Auerbach publications (2016)
63. Schaffer, J.D., Caruana, R.A., Eshelman, L.J.: Using genetic search to exploit the emergent behavior of neural networks. Physica D: Nonlinear Phenomena 42(1-3), 244–248 (1990)
64. Schiffmann, W., Joost, M., Werner, R.: Synthesis and performance analysis of multilayer neural network architectures (1992)
65. Schoenauer, M., Ronald, E.: Genetic extensions of neural net learning: Transfer functions and renormalisation coefficients
66. Sietsma, J., Dow, R.J.: Creating artificial neural networks that generalize. Neural Networks 4(1), 67 – 79 (1991)
67. Silva, S., Ingalalli, V., Vinga, S., Carreiras, J.M., Melo, J.B., Castelli, M., Vanneschi, L., Gonçalves, I., Caldas, J.: Prediction of forest aboveground biomass: an exercise on avoiding overfitting. In: European Conference on the Applications of Evolutionary Computation, pp. 407–417. Springer (2013)
68. Srinivas, M., Patnaik, L.M.: Learning neural network weights using genetic algorithms-improving performance by search-space reduction. In: [Proceedings] 1991 IEEE International Joint Conference on Neural Networks, vol. 3, pp. 2331–2336 (1991)
69. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence 1(1), 24–35 (2019)
70. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 10(2), 99–127 (2002)
71. Sutton, R.S.: Two problems with backpropagation and other steepest-descent learning procedures for networks. In: Proceedings of the Eighth Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum (1986)
72. White, D., Ligomenides, P.: Gannet: A genetic algorithm for optimizing topology and weights in neural network design. In: International Workshop on Artificial Neural Networks, pp. 322–327. Springer (1993)
73. Whitley, D., Starkweather, T., Bogart, C.: Genetic algorithms and neural networks: Optimizing connections and connectivity. Parallel computing 14(3), 347–361 (1990)
74. Widrow, B., Lehr, M.A.: 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. Proceedings of the IEEE 78(9), 1415–1442 (1990)
75. Wilson, S.W.: Perception redux: Emergence of structure. Physica D: Nonlinear Phenomena 42(1-3), 249–256 (1990)
76. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE 87(9), 1423–1447 (1999)
77. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. IEEE transactions on neural networks 8(3), 694–713 (1997)
78. Zhang, C., Shao, H., Li, Y.: Particle swarm optimisation for evolving artificial neural network. In: Systems, Man, and Cybernetics, 2000 IEEE International Conference on, vol. 4, pp. 2487–2490. IEEE (2000)
79. Zhang, G., Patuwo, B.E., Hu, M.Y.: Forecasting with artificial neural networks:: The state of the art. International journal of forecasting 14(1), 35–62 (1998)

View publication stats