A Greedy Iterative Layered Framework for Training Feed Forward Neural Networks

L. L. Custode[5], C. L. Tecce[1], I. Bakurov[3], M. Castelli[4], A. Della Cioppa[1,2](✉), and L. Vanneschi[3,4](✉)

[1] Natural Computation Lab, DIEM, University of Salerno, Fisciano, Italy
adellacioppa@unisa.it
[2] ICAR-CNR, Via P. Castellino, 111, 80131 Naples, Italy
[3] NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisbon, Portugal
lvanneschi@novaims.unl.pt
[4] LASIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisbon, Portugal
[5] Department of Information Engineering and Computer Science, University of Trento, Trento, Italy

Abstract. In recent years neuroevolution has become a dynamic and rapidly growing research field. Interest in this discipline is motivated by the need to create ad-hoc networks, the topology and parameters of which are optimized, according to the particular problem at hand. Although neuroevolution-based techniques can contribute fundamentally to improving the performance of artificial neural networks (ANNs), they present a drawback, related to the massive amount of computational resources needed. This paper proposes a novel population-based framework, aimed at finding the optimal set of synaptic weights for ANNs. The proposed method partitions the weights of a given network and, using an optimization heuristic, trains one layer at each step while "freezing" the remaining weights. In the experimental study, particle swarm optimization (PSO) was used as the underlying optimizer within the framework and its performance was compared against the standard training (i.e., training that considers the whole set of weights) of the network with PSO and the backward propagation of the errors (backpropagation). Results show that the subsequent training of sub-spaces reduces training time, achieves better generalizability, and leads to the exhibition of smaller variance in the architectural aspects of the network.

Keywords: Neuroevolution · Particle swarm optimization · Artificial neural networks

1 Introduction

An artificial neural network (ANN) is a biologically-inspired computer system that simulates the biological neural networks (BNNs) and their biochemical processes [10]. As such, an ANN consists of a set of interconnected layers of basic processing units, called neurons, which, altogether, comprise a powerful and versatile computing system that can solve numerous complex problems, including, but not limited to automatic speech and image recognition, natural language processing, autonomous car driving, bio-informatics, fraud-detection [24].

Several properties of ANNs make them an appropriate technique to solve supervised machine-learning (SML) problems. First, ANNs are data-driven, self-adaptive, real-time learning methods, that rely on few *a priori* assumptions about the models for problems under study [23,33]. Thus, the ANNs are suitable for problems with solutions requiring knowledge that may be difficult to specify, but for which there are sufficient data to do so. Second, ANNs are universal function approximators. More specifically, ANNs have been proven to be capable, with some limitations, of approximating any continuous function to any desired level of accuracy [12–14]. The effectiveness of ANNs derives from the neurons' mode of interacting with each other, i.e. their interconnections. Traditionally, such interconnections are assigned random weights to initialize a network. Then, a training algorithm is executed to identify the optimal set of weights, thereby allowing the network to achieve *satisfactory* performance, given the problem at hand. Typically, within the context of SML, such a training algorithm consists of the backward propagation of the errors, known as *backpropagation.* This is an optimization algorithm, based on the gradient of a loss function in the weight space [22,31].

Backpropagation is one of the most popular training algorithms for ANNs. In practice, however, backpropagation presents significant limitations related to multi-modal and non-differentiable cost functions. Moreover, backpropagation is particularly inefficient and time-consuming in the face of problems with a huge number of local optima and *plateaus* in the error surface [11].

In the context of deep learning (DL), the backpropagation is known to suffer from vanishing/exploding gradients. This phenomenon refers to a situation in which the gradient of a cost function can become exceedingly large or extremely small. More specifically, the backpropagation performs very small/big updates to the set of weights, thereby rendering the training process imprecise and time-consuming [15].

This work introduces a novel population-based framework for training ANNs. The proposed technique optimizes the weights of an ANN, one layer per step, while freezing all remaining weights in the other layers. This approach forces the optimization algorithm to work in a subspace of the original search space. This choice boasts two main advantages compared with the more traditional approach, which is aimed at optimizing the whole network at once: (i) it increases the effectiveness of the exploration and (ii) reduces the time needed for the optimization process. To investigate the capability of the technique proposed in this work, particle swarm optimization (PSO) was used as the underlying optimizer in our experiments. However, the framework is flexible to a point, to allow for the use of any optimization heuristic. In the following discussion, we present results, obtained on a set of well-known benchmark problems, which demonstrate the capacity of the proposed system to produce results better than (or comparable to) those yielded by backpropagation and to produce them more quickly, in terms of training time, than traditional approaches.

It is important to note here that the main purpose of this paper is to present our preliminary results concerning the ability of the proposed layered paradigm to optimize an ANN, but that there has been no special effort expended to identify the most suitable optimizer for the problem at hand. For this reason, only standard versions of the PSO and backpropagation algorithms have been used, despite the existence of several well-known variants of these algorithms that may be able to achieve better performance.

For the same reason, only feed-forward fully-connected networks have been considered in the performed experiments, even if, in principle, the proposed technique applies to recurrent networks as well.

The paper is organized as follows: Sect. 2 reviews prior related work, aimed at defining techniques for optimizing the weights of ANNs; Sect. 3 describes the proposed approach; while Sect. 4 presents the experimental settings and discusses the obtained results; and finally, Sect. 5 summarizes the main findings of this paper and suggests ideas for future research.

2 Prior Related Work

It is challenging to train an ANN to find an optimal, or at least a satisfactory, set of synaptic weights for a given problem. We refer to this task as WEANN (Weight Evolving Artificial Neural Networks), which is a simplified version of the standard Neuroevolution problem. The difficulty is associated mainly with the non-linearity of the problem that the network is meant to solve, the lack of knowledge regarding the best set of weights and biases, and the dependency of the performance of the training algorithm on the architectural aspects of the network (specifically, the topology and activation functions of the neurons). Hence, given their utility as alternatives to backpropagation and its variants [21], heuristic search algorithms were used to optimize ANNs [3,18,26,28].

Within this research track, evolutionary algorithms (EAs) were assigned a special role, due to their recognized advantages over gradient-based techniques [23,31]. EAs are better at handling a global search in a vast, complex, multimodal and non-differentiable surface, and especially useful when the gradient of the cost function is expensive to calculate. Moreover, EAs can be used to train many types of ANNs, including feed-forward, recurrent and higher-order. Finally, EAs can easily incorporate special characteristics, such as regularization. Among the many existing references that report on the use of EAs to optimize ANNs, we refer the reader to [23,31].

Given our selection of PSO as the underlying optimization algorithm to conduct our experiments, in this section, we focus on prior contributions studying the application of PSO to training ANNs. In [9,16,32] PSO was used to evolve the weights and topology of the network. In particular, in [16], authors presented a multi-dimensional particle swarm optimization (MD-PSO) technique to automatically design ANNs through a process of evolution to the optimal network configuration (consisting of connections, weights, and biases) within an architecture space. Similarly, in [9], the study consisted of performing a simultaneous evolution of an ANN's three principal components: the set of synaptic weights, the architecture, and the transfer functions of each neuron. The main topic of the paper was to find the optimal design of an ANN, using eight proposed fitness functions to evaluate the quality of each solution. In [32], the authors introduced a new evolutionary system, constrained to the use of PSO, for feed-forward ANNs. The architecture and the weights of ANNs were adjusted adaptively according to the quality of the network.

All of these approaches differ substantially from the one presented in this paper, because their performance is not directly comparable with standard training algorithms, such as backpropagation. In fact, the objective of standard training algorithms is "only" to find an optimal set of synaptic weights, and that is also the objective of the system presented here. Another chief and distinctive feature of this work is its lack of *apriori* considerations regarding the extent to which the standard PSO was effective at training an ANN to solve a given optimization problem. In this paper, we show it is possible to achieve a better performance by reducing the search space by constraining the evolutionary process within a subspace of the whole solution space, without reducing the exploratory power of the selected optimization technique.

In [1] and [25], the authors applied PSO in the context of using cooperative learning (CL) to train ANNs. The training process was conducted within a multi-swarm architecture, such that each swarm optimized a subset of the solution's vector, which was previously split and distributed across swarms according to a given rule. To identify an optimal solution to the original problem, it was necessary to combine particles from all swarms to form the candidate solution vector, which was then passed to the fitness function. In [1], the authors presented three splitting rules to train ANNs for classification problems. Each time a new global best solution was found, the fitness value of the best particles in all the swarms was updated to reflect that value, because of all these particles participated in the creation of the global best. A similar idea was proposed in [25], in which the authors sought to improve the performance of the basic PSO by halving the solution vector (Esplit) and allocating each on its swarm. Then, a plain swarm, containing the entire solution vector, was used as an "attractor" for other two swarms. Both approaches suffer from problems related to selection of which particles to bind for composition of the solution, and the assignment of fitness within the evolutionary process. Regarding the latter, the question discussed in [1] is: How much credit should each swarm be awarded when the combined vector (constructed from all the swarms) results in a better solution? It is not, in fact, an easy task, to define a proper way to split the value of the fitness so as to give the bulk of it to the best swarm involved in representing the final solution. A precise answer to this question, supported by experimental results, was not provided.

Given the nomenclature assigned to the CL framework, the technique presented in this paper uses Lsplit (or Layer-split) architecture to distribute the network's weights across different swarms. However, unlike the approaches described above, the swarms are not evolved simultaneously, but rather sequentially, one after another, layer after layer.

Our approach does not suffer from any of the aforementioned problems, because the optimization phase of a given layer is constrained to the remaining part of the network. In fact, the main objective is to maximize the performance of the whole network by adapting the behavior of each layer to the ones that follow.

In [34], the authors presented a hybrid algorithm (called PSO-BP), combining PSO with backpropagation, to train the weights of feedforward ANNs. The main idea is to combine the rapid converge of PSO during the initial stages of a global search (as the PSO search process is likely to slow considerably as it approaches a global optimum) with the gradient descent method that can achieve faster convergence around global optima. The proposed PSO-BP algorithm presents a heuristic way to give a transition from particle swarm search to gradient descent. The experimental results showed that the PSO-BP algorithm is better than the backpropagation algorithm in terms of convergence speed and accuracy.

3 The Proposed Method

Due to the nature of gradient descent optimization, the backpropagation suffers from certain drawbacks that can, in some cases, compromise its effectiveness. For instance, the backpropagation can get stuck in local optima or have problems optimizing error surfaces with vast neutral *plateaus*. Furthermore, non-differentiability and vanishing/exploding gradients can be other possible sources of difficulties. These problems have been solved partially by stochastic gradient descent with momentum [19], but they remain an issue.

Evolutionary ANNs (EANNs) rely on the use of evolutionary and nature-inspired algorithms and can represent an alternative to gradient-based techniques. Although EANNs are characterized by their adaptability, broad range of applications and flexibility, they usually require significant amounts of computational power to perform the training task. Considering a population-based optimization heuristic and that each individual in the population represents the whole set of synaptic weights, as the network grows bigger and deeper, so increases the computational power required to train the system [27,30].

The purpose of the algorithm proposed in this paper is to use a population-based heuristic to overcome the exploratory shortcomings of the backpropagation, while reducing the computational effort required during training of ANNs. The proposed system is characterized by two main elements: (i) it is a population-based heuristic algorithm, which allows us to conduct a global optimization in a potentially huge, rugged, multi-modal and non-differentiable search space, avoiding *plateaus* in the error surface and eliminating the vanishing/exploding gradient problem; and (ii) considering a n-dimensional search space, where each dimension represents a weight in a given ANN, and a candidate solution is an n-dimensional vector of candidate weights, the algorithm severs the layers, splitting the search space into subspaces that are then optimized sequentially.

In recent years, deep learning has become very popular and state-of-the-art networks usually contain many layers. Therefore, to reduce the computational time needed for their optimization, it makes sense to avoid using population-based heuristics to achieve a simultaneous optimization of the whole network. For this reason, we considered splitting the solution vector into semantically different subsets of the search space, each optimized within its population.

Our solution to the problem of ANN weights optimization is called greedy iterative layered training (GILT) algorithm and is described in the next section. It is worth noting here that, in theory, GILT can use any optimization heuristic. However, given that, in our experiments, we have chosen to use PSO as the optimization algorithm for GILT, the method will hereinafter be referred to as GILT-PSO.

3.1 Greedy Iterative Layered Training

The GILT algorithm conducts a greedy training of an ANN using an arbitrary population-based heuristic algorithm. By greedy, we mean that the algorithm does not seek to train the whole network at the same time. Instead, it searches, one layer at a time, for an optimal set of weights, while freezing the other layers. Each layer is optimized in its turn, and the whole procedure is iterated until the satisfaction of a stopping criterion or through a predefined number of epochs. During the optimization of a given layer, the algorithm's objective is to find a set of weights for the current layer that maximizes the performance of the whole network. In GILT, after the optimization of a given layer, the final population is stored for later use in seeding the new population that will optimize the same layer in the subsequent epoch. In other words, throughout this framework, a layer can be viewed as a point in an n-dimensional space and, at the end of the optimization process, we will have not only one single point representing the layer, but for each layer we will have a cloud of available points. If a sufficient number of iterations is given to the optimization technique, even after the first epoch, the population optimizing each layer is likely to reach a good degree of convergence. Thus, it is expected that the populations will be made of the best individual and a certain number of its perturbations. We expect this training framework to provide higher exploratory power than the backpropagation, while maintaining good exploitability. At the same time, semantically splitting the solution space by ANN layers forces the heuristic algorithm to conduct the optimization within a subspace of the search space, which is unlike the simultaneous optimization of all layers. In conclusion, it is important to note that the final solution is constrained to the initial setting of the network because all adjustments to the weights are influenced by the way the neurons in the other layers perceive them. Moreover, repeating this process for more than one epoch only serves to fine-tune the solution obtained on the first one. It is worth noting that our methodology does not make use of minibatches, which means that, for each

Algorithm 1. Network Training

```
Input  : $O$ - Optimizer, $l$ - Layout, $X$ - Training set, $y$ - Training labels,
         $max\_epochs$ - Number of epochs,
         $stopping\_criterion$ - A stopping criterion method
Output: $w$ - Set of weights for the whole network that minimizes loss
1  $current\_epoch \leftarrow 0$
2  $w \leftarrow Random()$
3  $seeds \leftarrow$ 2-dimensional array with initial weights
4  while $current\_epoch < max\_epochs$ and not $stopping\_criterion()$ do
5      foreach $(l\_index, layer)$ in $w$ do
6          $layer \leftarrow O.optimize\ (\ seeds[l\_index],\ l\_index,\ w,\ X,\ y\ )$
7          $seeds[l\_index] \leftarrow O.get\_population()$
8          $w \leftarrow replace(w, l\_index, layer)$
9      end
10     $current\_epoch \leftarrow current\_epoch + 1$
11 end
12 return $w$
```

fitness evaluation, all contents of the dataset are fed to the ANN. The mean squared error between the expected output and the current one has been used as a fitness metric. The proposed method is described in Algorithm 1.

4 Experimental Study

4.1 Test Problems

The performance of GILT-PSO has been tested on four different classification problems and with several network topologies for each problem. The reason behind this choice is to prove the general suitability of GILT-PSO; thus, we preferred to test the algorithm under different (although simple) test cases instead of focusing on a single complex task.

The datasets used have been taken from the UCI repository [5] and they are:

1. **Iris:** This dataset consists of measurements related to three classes of the iris plant, and it contains 50 instances for each class. One class is linearly separable from the other two; the latter is not. Each sample is described in terms of four features: sepal length in cm, sepal width in cm, petal length in cm, petal width in cm [8].
2. **Seeds:** Data belonging to three varieties of wheat: Kama, Rosa, and Canadian; 70 elements of each, randomly selected for the experiment. High-quality visualization of the internal kernel structure was detected, using a soft x-ray technique. The images were recorded on 13×18 cm x-ray KODAK plates. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin. Each sample is described in terms of seven

features: area, perimeter, compactness, length of the kernel, width of the kernel, asymmetry coefficient, length of the kernel groove [2].
3. **Breast Cancer Wisconsin (Diagnostic):** Features are computed from a digital image of fine needle aspirate (FNA) of a breast mass. They refer to characteristics of the cell nuclei that are visible in the image. The dataset consists of two classes (malignant and benign) and 569 total samples. Each sample is described in terms of 30 features: mean, standard error and worst of radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimension [17,29].
4. **Wine:** Results of a chemical analysis of wines grown in the same region in Italy, but derived from three cultivates. The dataset is characterized by three classes and 178 total observations. Each sample is described in terms of 13 features: alcohol, malic acid, ash, alkalinity of ash, magnesium, total phenols, flavonoids, non-flavonoid phenols, proanthocyanidins, color intensity, hue, OD280/OD315 of diluted wines, proline.

4.2 Experimental Setting

For each of the above datasets, several topologies for the hidden layers have been used, while the number of input and output neurons are set according to the number of features and classes for the specific problem. The ANNs used are fully connected and they present the following topologies: (in, 3, out), (in, 20, out), (in, 30, 20, out), (in, 50, 30, out), (in, 40, 40, 30, out), where *in* and *out* stand respectively for the number of input features and target classes of the underlying problem. All hidden neurons have the same activation function, which is the hyperbolic tangent, while the output layer's activation is Softmax.

The results obtained using GILT-PSO have been compared with the ones obtained by the standard backpropagation algorithm, and implemented in the SciKit framework [20] through the class MLPClassifier. The parameters have been chosen after a coarse-grained grid search and are shown below:

- activation = 'tanh': Hyperbolic tangent
- solver = 'sgd': Stochastic gradient descent
- alpha = 0: L2 regularization parameter
- shuffle = True: Shuffling of the dataset for each epoch
- learning_rate = 'constant'
- learning_rate_init = 0.03
- validation_fraction = 0
- early_stopping = False
- tol = 0: Used for the early stopping
- momentum = 0.9
- batch_size = size of the training set

It should be noted here that using a batch size equal to the entire dataset "guarantees" the convergence for the MLP classifier, thereby complicating the comparison with respect to GILT-PSO.

To offer empirical proof of the effectiveness of GILT-PSO, we performed a comparison with the standard PSO; in this context, we use the term "standard PSO" to indicate a PSO with the objective of optimizing all the weights in the ANN at once, i.e. without splitting them into layers. The hyperparameters for PSO and GILT-PSO are the following: $w = 0.73$, $c_1 = 1.5$, $c_2 = 1.5$, and they have been chosen according to [6]. The version of the PSO algorithm used is the classical one described in [7].

Because this work aims to prove the effectiveness of the proposed layered approach without depending on low-level details of the underlying optimizers, we decided to use basic variants of both backpropagation and PSO.

During the experiments, the goal was to perform a fair evaluation across all algorithms under investigation; thus, the number of epochs for the backpropagation was calculated according to the following formula:

$$n_{\text{BPepochs}} = n_{\text{particles}} \cdot n_{\text{iterations}} \cdot n_{\text{epochs}} \cdot (n_{\text{hidden layers}} + 1)$$

where $n_{\text{hidden layers}}$ represents the number of hidden layers of a given network. However, for the standard PSO, the number of iterations was calculated by:

$$n_{\text{pso\_it}} = n_{\text{iterations}} \cdot n_{\text{epochs}} \cdot (n_{\text{hidden layers}} + 1)$$

These equations ensure the same number of fitness evaluations for all the algorithms under comparison.

The major objective of our experiments was to compare the performance of different training algorithms for different ANN architectures across different problems. For each layout and each dataset, 30 independent runs were performed and final classification accuracy on the training and validation set, for each network, was stored. The validation accuracy was then used to select the best network in order to evaluate it on the test set. It is worth to note that, for each run, a new set of random weights are generated and used as a starting point for all the algorithms under comparison. Based on these results, we performed a statistical analysis to validate our conclusions. For the first experiment, the used parameters were the following: number of epochs equal to 2, number of iterations equal to 17 and number of particles equal to 15.

Using only two epochs was found to be sufficient to evaluate the performance of the network after a first rough optimization by GILT-PSO and, thus, to understand its effectiveness.

4.3 Experimental Results

Table 1 reports a broad perspective of the experimental results that we obtained using 2 epochs. Here, the reader can find the average classification accuracy on the validation set, and the respective standard deviation, for each training algorithm, calculated using the best solutions found (in terms of training accuracy), at the end of each run, across all studied problems and topologies. From the table, the competitive advantage of GILT-PSO over backpropagation and standard PSO is clear: GILT-PSO is not only, in average terms, more accurate, but also more stable (in fact, it has a lower standard deviation).

Table 1. Summary of the validation accuracy in terms of mean and standard deviation across all the problems under examination.

Algorithm	Mean	Std deviation
BP	0.889	0.096
GILT-PSO	0.903	0.034
PSO	0.898	0.040

Table 2. Median accuracy on the test set of the best networks evaluated on the validation set - 2 epochs.

Dataset	Layout	BP	GILT-PSO	PSO
Iris	[4, 3, 3]	0.889	**0.944**	0.861
	[4, 20, 3]	**0.889**	0.833	0.833
	[4, 30, 20, 3]	**0.889**	**0.889**	**0.889**
	[4, 50, 30, 3]	**0.889**	**0.889**	**0.889**
	[4, 40, 40, 30, 3]	**0.944**	0.889	0.889
Seeds	[7, 3, 3]	**0.913**	0.870	**0.913**
	[7, 20, 3]	0.870	**0.913**	0.826
	[7, 30, 20, 3]	0.870	0.870	**0.913**
	[7, 50, 30, 3]	0.739	**0.913**	**0.913**
	[7, 40, 40, 30, 3]	0.652	**0.957**	0.870
Wine	[13, 3, 3]	0.824	0.853	**0.882**
	[13, 20, 3]	0.824	0.882	**0.941**
	[13, 30, 20, 3]	**0.882**	**0.882**	0.706
	[13, 50, 30, 3]	0.824	0.824	**0.912**
	[13, 40, 40, 30, 3]	**0.882**	**0.882**	0.853
Breast	[30, 3, 2]	0.482	0.786	**0.804**
	[30, 20, 2]	0.482	**0.875**	0.795
	[30, 30, 20, 2]	0.411	**0.696**	0.482
	[30, 50, 30, 2]	0.411	**0.688**	0.607
	[30, 40, 40, 30, 2]	0.634	0.554	**0.732**

If we study the results obtained by the different algorithms on each of the studied problems, we can see that, in the majority of the cases, GILT-PSO outperforms both backpropagation and PSO or returns a result comparable to the best of them. More specifically, when selecting the best solution (in terms of training accuracy) from the 30 runs, the highest validation accuracy, for most of the problems and topologies, is obtained using GILT-PSO. These results, considering 2 epochs, are summarized in Table 2, where the best accuracy values for each row are highlighted in bold.

Table 3. Median accuracy on the test set of the best networks evaluated on the validation set - 3 epochs.

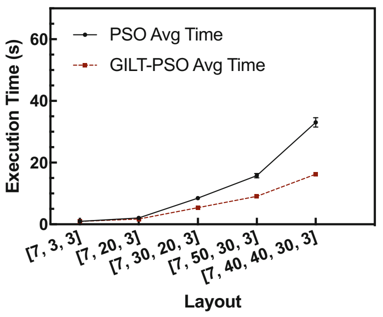
Dataset	Layout	BP	GILT-PSO	PSO
Iris	[4, 3, 3]	0.889	**0.917**	0.889
	[4, 20, 3]	**0.889**	0.833	**0.889**
	[4, 30, 20, 3]	**0.889**	**0.889**	**0.889**
	[4, 50, 30, 3]	**0.889**	**0.889**	**0.889**
	[4, 40, 40, 30, 3]	0.833	**0.889**	**0.889**
Seeds	[7, 3, 3]	**0.913**	0.870	0.826
	[7, 20, 3]	**0.913**	0.891	0.870
	[7, 30, 20, 3]	0.826	**0.913**	0.891
	[7, 50, 30, 3]	0.870	**0.913**	**0.913**
	[7, 40, 40, 30, 3]	0.739	**0.913**	0.870
Wine	[13, 3, 3]	0.824	0.882	**0.941**
	[13, 20, 3]	0.824	0.882	**0.912**
	[13, 30, 20, 3]	0.882	0.882	**0.941**
	[13, 50, 30, 3]	0.824	0.912	**0.941**
	[13, 40, 40, 30, 3]	**0.882**	**0.882**	0.765
Breast	[30, 3, 2]	0.464	**0.804**	0.661
	[30, 20, 2]	0.464	**0.661**	0.446
	[30, 30, 20, 2]	0.429	**0.804**	0.607
	[30, 50, 30, 2]	0.420	**0.536**	0.500
	[30, 40, 40, 30, 2]	0.643	**0.750**	0.634

Here, we can observe that, 11 times out of 20, GILT-PSO outperforms or furnishes comparable results to both backpropagation and standard PSO; more specifically, it outperforms the backpropagation 16 times.

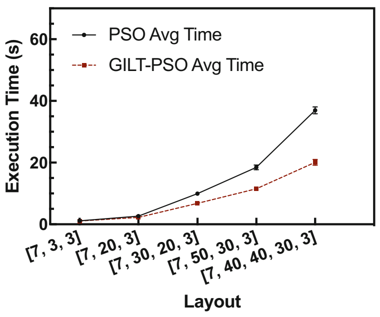
The same experiments were repeated for when the number of epochs was equal to three. The results are shown in Table 3. In this case, GILT-PSO outperforms the other algorithms or returns a result that is comparable to the best of them, in 13 cases out of 20.

4.4 Execution Time Comparison

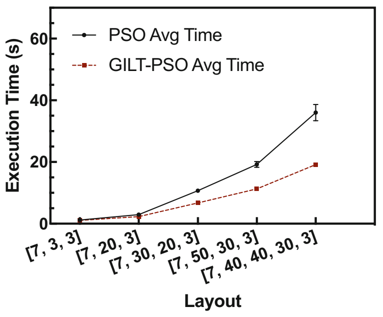
Another advantage of the layered approach used by GILT-PSO is its efficiency, in terms of the running time needed for the training phase. As already discussed, optimizing only one layer at a time allows the chosen optimizer to reduce the dimensionality of the search space. First, working with smaller candidate solutions reduces the computational time needed to perform the mathematical operations involved in the optimization process. Second, having smaller candidate solutions also means working with smaller amounts of data, thereby reducing the consequent overhead.



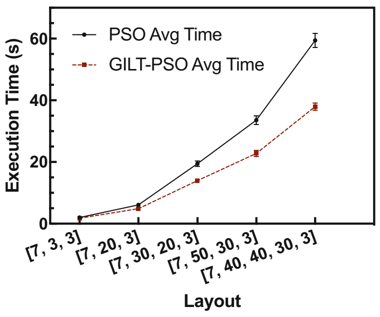
(a) Iris.



(b) Seeds.



(c) Wine.



(d) Breast Cancer.

Fig. 1. Comparison of the average execution times between the GILT-PSO algorithm and the PSO algorithm. The error bars account for the standard deviations.

It is worth to notice that only the running times regarding the standard PSO and GILT-PSO are presented in this paper because both algorithms rely on the framework which was created in the context of this research, i.e., we can guarantee the implementation compatibility for both PSO-based approaches, while the used implementation of the backpropagation relies on the external library [20]. However, GILT-PSO and PSO were the methods that returned the best results in terms of accuracy, as reported in the previous section. As such, it makes sense to compare those two systems as well, in terms of running time. The comparison is reported in Fig. 1. Based on this figure, it is clear that GILT-PSO is faster than PSO, and that the difference between these two methods grows increasingly visible as the size of the network increases.

4.5 Non-parametric Statistical Test with Control Method

To evaluate whether the previously presented results differ from each other in a statistically significant way, a statistical analysis, based on a non-parametric test [4] has been performed. In particular, given that the data are not normally

Table 4. Adjusted p-values for the post-hoc procedures for Aligned Friedman test (backpropagation is the control method) for 2 epochs.

Algorithm	p	z	Bonferroni	Holm	Holland	Rom	Finner	Li
GILT-PSO	0.056	1.915	0.056	0.025	0.025	0.025	0.025	0.032
PSO	0.385	0.869	0.385	0.050	0.050	0.050	0.050	0.050

Table 5. Adjusted p-values for the post-hoc procedures for Aligned Friedman test (backpropagation is the control method) for 3 epochs.

Algorithm	p	z	Bonferroni	Holm	Holland	Rom	Finner	Li
GILT-PSO	0.182	1.335	0.182	0.025	0.025	0.025	0.025	0.037
PSO	0.298	1.041	0.298	0.050	0.050	0.050	0.050	0.050

distributed, an aligned Friedman test was executed for multiple comparisons. The null hypothesis H_0 for this test states that the medians between the different algorithms are identical. The goal of the test is to confirm this hypothesis or reject it, with the level of confidence $\alpha = 0.05$. The p-value we have obtained applying the Aligned Friedman method is equal to 0.0005. This value is lower than the chosen level of confidence, which confirms the existence of statistically significant differences among the results returned by the studied algorithms. Given that H_0 is rejected, we can proceed with the *post-hoc* procedures to investigate whether the results returned by the PSO-based methods (PSO and GILT-PSO) are statistically different from the ones returned by the backpropagation. To this end, a new null hypothesis H_0' is used. H_0' states that the performance of a PSO-based algorithm is identical to the one of the backpropagation, which is used as a control method. Table 4 reports the adjusted p-values for the post-hoc procedures for Aligned Friedman test, in case the number of epochs is equal to 2. In the table, the generic (i, j) adjusted p-value represents the smallest level of significance that results in the rejection of H_0' between algorithm i and the control method j, i.e., the lowest value for which the algorithm i and the control method are not statistically equivalent, for the j-th *post-hoc* procedure. The lower a value, the more likely the null hypothesis can be rejected. A very important feature of such a table is that it is not tied to a pre-set level of significance α. Rather, depending on the value of α we choose, the *post-hoc* procedures will either accept or reject H_0'. Considering the value used for α, the table says that the statistical equivalence between backpropagation and GILT-PSO can be statistically rejected by all of the *post-hoc* procedures, except for Bonferroni since the adjusted p-values for those methods are lower than 0.05. Furthermore, the statistical equivalence between PSO and backpropagation cannot be rejected by all the procedures.

For the number of epochs equal to 3, the results of the *post-hoc* analysis are reported in Table 5. Considering the value used for α, this analysis suggests that the statistical equivalence between backpropagation and GILT-PSO can be rejected by all of the *post-hoc* procedures except for Bonferroni, as all the adjusted p-values for those methods are smaller than 0.05. Moreover, the statistical equivalence between PSO and backpropagation cannot be statistically rejected by all the procedures.

5 Conclusions

A new greedy optimization algorithm, called greedy iterative layered training (GILT), to optimize the synaptic weights of an artificial neural network (ANN), was proposed in this paper. GILT uses a population-based optimization heuristic and works by optimizing the weights by layers. At each step, the synaptic weights of one layer are optimized, without modifying the weights of all the other connections in the network. Given that in this study we used particle swarm optimization (PSO), the system was referred to as GILT-PSO.

According to the presented results, GILT-PSO outperforms backpropagation when considering the best networks trained among a set of 30 runs; it also outperforms PSO in terms of training time, while maintaining a comparable exploratory power. The presented statistical analysis shows that there is a significant difference between the results returned by GILT-PSO and the ones of backpropagation. Furthermore, the results returned by GILT-PSO in the different runs possess a smaller standard deviation than the ones of backpropagation, which seems to indicate that GILT-PSO may exhibit more stable (and thus more reliable) behavior.

These achievements are important and encourage us to pursue the study, particularly considering that GILT benefits from other potential advantages: it is not bounded to a specific optimization technique or fitness measure. In other words, in principle, it can be used not only with the PSO but also with any other population-based optimization heuristic. Moreover, when defining a new fitness measure and a new representation for the individuals, GILT can also be used to evolve the structure of the network as well as the neurons activation functions.

In the future, we plan to extend the study by using other types of optimization techniques, instead of PSO, and by extending the representation of the solutions and the fitness in such a way that GILT can be used to optimize the synaptic weights, the structure of the network and the activation functions of the neurons at the same time. Moreover, we plan to implement minibatch learning to increase the efficiency of our methodology. Furthermore, to better assess the capabilities and limitations of the proposed algorithm, more (and more complex) test cases must be considered.

Finally, we are currently working on the development of strategies to further improve the efficiency (in terms of training time) of the proposed algorithm. In the current version, in fact, to calculate fitness we need to perform a certain number of matrix operations, that involve the evaluation of the whole network.

But considering that at each step only one layer at a time can change, the process can be significantly optimized. In particular, each time that a layer is considered, it is possible to store the results of the partial evaluations of the previous layers. In this way, we could perform only the matrix operations of the current and following layers. It makes sense to expect that, particularly for deep networks, this will represent a considerable performance improvement.

Acknowledgements. This work was partially supported by FCT, Portugal through funding of LASIGE Research Unit (UID/CEC/00408/2019), and projects PREDICT (PTDC/CCI-CIF/29877/2017), BINDER (PTDC/CCI-INF/29168/2017), GADgET (DS-AIPA/DS/0022/2018) and AICE (DSAIPA/DS/0113/2019) and by the financial support from the Slovenian Research Agency (research core funding No. P5-0410).

This work is the result of the collaboration between the University of Salerno and Nova IMS. The first two authors contributed equally to this work.

References

1. Van den Bergh, F., Engelbrecht, A.P.: Cooperative learning in neural networks using particle swarm optimizers. South Afr. Comput. J. **2000**(26), 84–90 (2000)
2. Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P.A., Łukasik, S., Żak, S.: Complete gradient clustering algorithm for features analysis of x-ray images. In: Piętka, E., Kawa, J. (eds.) Information Technologies in Biomedicine. AINSC, vol. 69, pp. 15–24. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13105-9_2
3. De Falco, I., Cioppa, A.D., Natale, P., Tarantino, E.: Artificial neural networks optimization by means of evolutionary algorithms. In: Chawdhry, P.K., Roy, R., Pant, R.K. (eds.) Soft Computing in Engineering Design and Manufacturing, pp. 3–12. Springer, London (1998). https://doi.org/10.1007/978-1-4471-0427-8_1
4. Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol. Comput. **1**, 3–18 (2011)
5. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
6. Eberhart, Shi, Y.: Particle swarm optimization: developments, applications and resources. In: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), vol. 1, pp. 81–86, May 2001
7. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, MHS 1995, pp. 39–43, October 1995. https://doi.org/10.1109/MHS.1995.494215
8. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Ann. Eugenics **7**(7), 179–188 (1936)
9. Garro, B.A., Vázquez, R.A.: Designing artificial neural networks using particle swarm optimization algorithms. Comput. Intell. Neurosci. **2015**, 61 (2015)
10. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall PTR, Upper Saddle River (1994)
11. Hochreiter, S.: Untersuchungen zu dynamischen neuronalen netzen. Diploma, Technische Universität München **91**(1) (1991)

12. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**(2), 251–257 (1991)
13. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
14. Irie, B., Miyake, S.: Capabilities of three-layered perceptrons. In: IEEE International Conference on Neural Networks, vol. 1, p. 218 (1988)
15. Kolbusz, J., Rozycki, P., Wilamowski, B.M.: The study of architecture MLP with linear neurons in order to eliminate the "vanishing gradient" problem. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2017. LNCS (LNAI), vol. 10245, pp. 97–106. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59063-9_9
16. Kiranyaz, S., Ince, T., Yildirim, A., Gabbouj, M.: Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. Neural Netw. **22**(10), 1448–1462 (2009)
17. Mangasarian, O.L., Street, W.N., Wolberg, W.H.: Breast cancer diagnosis and prognosis via linear programming. Oper. Res. **43**(4), 570–577 (1995)
18. Montana, D.J., Davis, L.: Training feedforward neural networks using genetic algorithms. In: IJCAI, vol. 89, pp. 762–767 (1989)
19. Moreira, M., Fiesler, E.: Neural networks with adaptive learning rate and momentum terms. Idiap-RR Idiap-RR-04-1995, IDIAP, Martigny, Switzerland, October 1995
20. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
21. Rashid, T.: Make Your Own Neural Network. CreateSpace Independent Publishing Platform (2016)
22. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science (1985)
23. Ding, S., Li, H., Su, C., Yu, J., Jin, F.: Evolutionary artificial neural networks: a review. Artif. Intell. Rev. **39**, 251–260 (2013)
24. Samarasinghe, S.: Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition. Auerbach Publications (2016)
25. Settles, M., Rylander, B.: Neural network learning using particle swarm optimization. In: Advances in Information Science and Soft Computing, pp. 224–226 (2002)
26. Shaw, D., Kinsner, W.: Chaotic simulated annealing in multilayer feedforward networks. In: Canadian Conference on Electrical and Computer Engineering, vol. 1, pp. 265–269. IEEE (1996)
27. Sher, G.I.: Handbook of Neuroevolution Through Erlang. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-4463-3
28. Si, T., Hazra, S., Jana, N.D.: Artificial neural network training using differential evolutionary algorithm for classification. In: Satapathy, S.C., Avadhani, P.S., Abraham, A. (eds.) Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA). AINSC, vol. 132, pp. 769–778. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27443-5_88
29. Street, W.N., Wolberg, W.H., Mangasarian, O.L.: Nuclear feature extraction for breast tumor diagnosis. Proc. Soc. Photo-Opt. Inst. Eng. **1993** (1999)
30. Whitley, D., Starkweather, T., Bogart, C.: Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Comput. **14**, 347–361 (1990)
31. Yao, X.: Evolving artificial neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)

32. Zhang, C., Shao, H., Li, Y.: Particle swarm optimisation for evolving artificial neural network. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, pp. 2487–2490. IEEE (2000)
33. Zhang, G., Patuwo, B.E., Hu, M.Y.: Forecasting with artificial neural networks: the state of the art. Int. J. Forecast. **14**(1), 35–62 (1998)
34. Zhang, J.R., Zhang, J., Lok, T.M., Lyu, M.R.: A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training. Appl. Math. Comput. **185**(2), 1026–1037 (2007)