Pattern Recognition Letters 125 (2019) 373–380

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

A continuation approach for training Artificial Neural Networks with meta-heuristics

Jairo Rojas-Delgado [a,*], Rafael Trujillo-Rasúa [b], Rafael Bello [c]

[a] Universidad de las Ciencias Informáticas, Habana, Cuba
[b] IN Switch Solutions S.A., Montevideo, Uruguay
[c] Universidad Central de las Villas, Santa Clara, Cuba

ARTICLE INFO

Article history:
Received 8 January 2019
Available online 23 May 2019

MSC:
90C15
90C56
62M45

Keywords:
Continuation
Optimization
Neural-network

ABSTRACT

Artificial Neural Networks research field is among the areas of major activity in Artificial Intelligence. Training a neural network is an NP-hard optimization problem that presents several theoretical and computational limitations. In optimization, continuation refers to an homotopy transformation of the fitness function that is used to obtain simpler versions of such fitness function and improve convergence. In this paper we propose an approach for Artificial Neural Network training based on optimization by continuation and meta-heuristic algorithms. The goal is to reduce overall execution time of training without causing negative effects in accuracy. We use continuation together with Particle Swarm Optimization, Firefly Algorithm and Cuckoo Search for training neural networks on public benchmark datasets. The continuation variations of the studied meta-heuristic algorithms reduce execution time required to complete training in about 5–30% without statistically significant loss of accuracy when compared with standard variations of the meta-heuristics.

1. Introduction

In the latest years, we have seen the efforts in optimization field to solve very complex and challenging tasks. At the same time, the success in this area has led researchers and practitioners to address much larger instances and more difficult classes of problems, specially in meta-heuristic area [1,2]. Training a neural network is an NP-hard optimization problem [3] that usually involves thousands and millions of parameters.

Additionally, the fitness function of Artificial Neural Networks (ANNs) is highly non-convex [4] and presents a poor correspondence between its local and global structure [5]. Also, the number of local minima increases exponentially respect to the number of parameters of the network. In fact, such local minima appears to be saddle points instead of true minima [6] reducing the convergence speed of optimization, specially for gradient based training algorithms. In general, second-order methods have not obtained better results than first-order methods in this area [5]. These facts become even more problematic as we start using modern deep learning architectures.

ANNs are widely used in several applications of image recognition [7], signal processing [8], speech [9] and several other fields. Nevertheless, neural networks are playing a major role in the field of pattern recognition and learning representations from data.

The most popular algorithm for training is Stochastic Gradient Descent (SGD) which is a gradient based first order[1] method that has been proven to converge into local minima in the parameter space. In general, first and second order[2] optimization methods require the calculation of derivatives and Hessians for which Automatic Reverse Mode Differentiation (ARMD) have to be used [10]. The parallel implementation of such methods faces important challenges due to the inherent sequential nature of ARMD.

Recently, several meta-heuristic algorithms for optimization have been actively used in ANN training. Meta-heuristic algorithms have features to escape local minima and increase the probability of global convergence. In the literature there are reports of nature-inspired meta-heuristics such as Cuckoo Search [11,12], Firefly Algorithm [13,14], Wolf Search Optimization [15,16], Particle Swarm Optimization (PSO) [17,18] and others.

For current applications, the rule of thumb to obtain higher accuracy have been to increase not only the number of parameters of ANNs but also the amount of training patterns mainly

[1] First order method refers to an optimization algorithm that needs information of the first derivative of the fitness function.
[2] Second order method refers to an optimization algorithm that needs information of the second derivative of the fitness function.

* Corresponding author.
E-mail address: jrdelgado@uci.cu (J. Rojas-Delgado).

https://doi.org/10.1016/j.patrec.2019.05.017
0167-8655/

because over-fitting [5,19]. In this scenario, the use of meta-heuristic algorithms is limited by the increased computational complexity of the fitness function evaluation. Additionally, state-of-the-art meta-heuristic algorithms are based in populations or swarms. This means, that we need to evaluate the fitness function for each solution in the population. Hence, a reduction in the computational complexity of the fitness function is of paramount importance.

In optimization field, continuation method refers to the general approach of start solving an easy problem and change it progressively during the course of iterations to the actual problem [20]. The way the problem is transformed from an easy version to the actual problem is usually done via homotopy [21]. Despite the skepticism for the case of non-convex optimization, this helps to reduce the probability of local minima convergence and reduce execution time of optimization [5].

In this work, we introduce an approach for ANNs training based on optimization by continuation and swarm-based meta-heuristic algorithms. First, we present a simple theoretical understanding of continuation to show the simplification of the error surface for an ANN. Further experimental results show that our optimization by continuation approach reduces execution time required to perform training in about $5\% - 30\%$ without statistically significant loss of accuracy.

The paper is structured as follows: Section 2 gives an overview on related work and Section 3 presents some insights on continuation methods. Next, we introduce an algorithm to train ANNs based on optimization by continuation and meta-heuristics. In Section 4 we discuss about accuracy and execution time results. Finally, some conclusions and recommendations are given. Throughout this article we use x for scalars, $\boldsymbol{x}$ for vectors and X for sets.

2. Related work

Continuation, embedding or homotopy methods have long served as useful tools in modern mathematics. For a complete and formal definition of continuation, please refer to [21]. For the sake of our discussion, considering a fitness function $f : \mathbb{R}^n \to \mathbb{R}^m$, we call homotopy or continuation of f to the deformation $H : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ such that:

$$H(\boldsymbol{x}, 1) = g(\boldsymbol{x}), H(\boldsymbol{x}, 0) = f(x) \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ is a trivial smooth function, usually convex.

There are examples of similar methods to continuation in machine learning and optimization in the literature. In [22] authors present an iterative method to add new neurons to an ANN during the course of optimization. Whenever a few epochs of optimization are completed, authors modify the ANN structure by adding new neurons to the hidden layer. This way, they increase the complexity of the optimization surface and the capacity of the model. In [23] authors use a fitness function transformation referred as function stretching. Function stretching consists of a two stage transformation of the fitness function that eliminates local minima, while preserving global ones. These two methods can be seen as a form of continuation.

A similar idea to continuation is incremental learning [24], however, the general idea is slightly different. In incremental learning we train models based on a earlier model and a new batch of data. Incremental learning mainly refers to learning from streaming data, which arrive over time, with limited memory resources and, ideally, without sacrificing model accuracy [25]. Studies in this area are mostly concentrated on issues like concept-drifting and catastrophic-forgetting [26,27].

In [28] the authors introduce an extension to mini-batch training with improved sampling strategies instead of shuffle. In this work, the size of the mini-batch increases over the course of iterations, changing effectively the complexity of the following optimization surfaces. However, this approach is 8.4% worst than regular mini-batch training, which is not acceptable for practical applications.

Continuation is also similar to a technique referred as surrogate function swindle [29]. In [29], authors suggest using surrogate functions that are faster to evaluate but not as accurate as the fitness function. According to [1] there are many problems for which the true fitness function is quite costly to evaluate. An effective approach to handle this issue is to evaluate neighbors using a surrogate, a function that is correlated to the true fitness function, but is less computationally demanding, see [30,31].

The Variable Neighborhood Search (VNS) heuristic [32] also shares some features with continuation. In VNS the neighborhood operator changes when a local minimum is found. This mean, that the shape of the optimization surfaces changes to a coarse-grained representation every time that optimization gets trapped in a local minima. After optimization is in a descending path, the algorithm changes again the neighborhood operator to a fine-grained representation of the optimization surface.

Continuation can be seen as a generalization of several methods that have been used for long in machine learning and optimization. In the next section we will introduce an approach for building an homotopy transformation that allows to take advantage of the efficiency of surrogate functions and the ability to escape from local minima.

3. Optimization by continuation

A fitness function evaluates the quality of a solution in an optimization problem. Its function is usually to find out what are the best solutions in a population. Intuitively, we do not need a fine-grained quality estimation of a solution to discriminate candidate solutions.

In practice the fitness function of an ANN is a sort of loss function such as the Mean Squared Error (MSE) in Eq. (2), where $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p\}, \boldsymbol{x}_i \in \mathbb{R}^d$ is the set of training patterns, $\boldsymbol{w} \in \mathbb{R}^n$ contains the neural network parameters and $\hat{y}_i$ and y_i are the expected and real output of the network for the $\boldsymbol{x}_i$ training pattern respectively.

$$f(\boldsymbol{w}, X) = \frac{\sum_{i=1}^{p} (\hat{y}_i - y_i)^2}{p} \tag{2}$$

For simplicity, here we only consider single-output regression problems. However, the following applies to any fitness function that holds the property of consistency [5], that is, as the number of data points p in the dataset increases, the estimate y_i converges to the true value $\hat{y}_i$. Consistency ensures that the bias induced by an estimator diminishes as the number of training patterns grows.

In Eq. (2) the computational cost comes from calculating the real output y_i that requires propagating a training pattern through each layer of the neural network. Actually, minimizing $f(\boldsymbol{w}, X)$ is an NP-hard optimization problem [3]. Eq. (3) formally defines ANN training as an optimization problem:

$$\boldsymbol{w} = \underset{\boldsymbol{w} \in \mathbb{R}^n}{\operatorname{argmin}} f(\boldsymbol{w}, X) \tag{3}$$

Having an ANN with two parameters or weights, its parameter space would look like a landscape with valleys and hills. As our goal is to find a point in such parameter space near of the global optimum, it is possible to use a coarse-grained representation of the error surface generated by the training patterns. Ideally, such error surface will have the most prominent irregularities in its topology, at least from a global point of view.

The former principle has been explored extensively and is commonly known in optimization as continuation method. The general

idea behind the so-called continuation method heuristic, is to start solving an easy problem and progressively change it to the actual complex task [20]. For clarification, our major issue here is how to find a coarse-grained representation of the error surface generated by presenting training patterns to an ANN.

Formally, given $\hat{X} \subset X$ and $\boldsymbol{w} \in \mathbb{R}^n$ it follows that: $f(\boldsymbol{w}, X) - \epsilon \leq f(\boldsymbol{w}, \hat{X}) \leq f(\boldsymbol{w}, X) + \epsilon$. The value of ϵ defines a property associated to $\hat{X}$ that we call intolerance:

Definition 1. Intolerance, is the value of ϵ associated to a subset of training patters $\hat{X}$ that measures its ability to approximate the optimization surface defined by the entire training set X.

Considering two vectors of parameters $\boldsymbol{w}_i, \boldsymbol{w}_j \in \mathbb{R}^n$ that represents two ANNs parameters, the following theorem holds:

Theorem 1. *If* $f(\boldsymbol{w}_i, \hat{X}) - f(\boldsymbol{w}_j, \hat{X}) > 2\epsilon$ *then* $f(\boldsymbol{w}_i, X) > f(\boldsymbol{w}_j, X)$.

Proof.

1. $f(\boldsymbol{w}_i, \hat{X}) - f(\boldsymbol{w}_j, \hat{X}) > 2\epsilon$ (Axiom)
2. $f(\boldsymbol{w}_i, X) + \epsilon - (f(\boldsymbol{w}_j, X) - \epsilon) > 2\epsilon$ (By definition of intolerance)
3. $f(\boldsymbol{w}_i, X) > f(\boldsymbol{w}_j, X)$ □

Theorem 1 enables the possibility that a subset $\hat{X}$ can be used instead X to discriminate between two vectors of parameters $\boldsymbol{w}_i$ and $\boldsymbol{w}_j$ with a fixed quote of intolerance ϵ. It is also possible to obtain an arbitrary amount $k < |\mathcal{P}(X)|$ of subsets $\hat{X}_i$ associated to X. Each $\hat{X}_i$ describes an optimization surface defined by the network parameters with an intolerance of ϵ_i.

Definition 2. We call a representative sequence of sets of training patterns to the sequence formed by $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$ such as $\epsilon_i \geq \epsilon_{i+1}$.

Definition 3. We call a restricted sequence of sets of training patterns to the sequence formed by $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$ such as $\hat{X}_i \subset \hat{X}_{i+1}$.

In practice, finding ϵ involves to solve Eq. (4). However, considering Theorem (2) is quite simple to obtain a representative sequence of sets of training patterns from a restricted sequence. The difficulty strives in using Theorem 1 because we need to know the value of ϵ associated to $\hat{X}$.

$$\epsilon = \max_{\boldsymbol{w} \in \mathbb{R}^n} |f(\boldsymbol{w}, X) - f(\boldsymbol{w}, \hat{X})| \tag{4}$$

Theorem 2. *If a sequence of sets of training patterns is restricted, then is also representative.*

Proof. From previous definitions, the value of ϵ can be seen as a sort of error that measures the difference between $f(\boldsymbol{w}, \hat{X})$ respect $f(\boldsymbol{w}, X)$. Then we have:

1. $f(\boldsymbol{w}, \hat{X}_i) \leq f(\boldsymbol{w}, X) + \epsilon_i$ (Axiom)
2. $f(\boldsymbol{w}, \hat{X}_{i+1}) \leq f(\boldsymbol{w}, X) + \epsilon_{i+1}$ (Axiom)
   by definition, we also know that
3. $|\hat{X}_i| < |\hat{X}_{i+1}|$ (By definition of restricted sequence)
   Considering the Vapnik–Chervonenkis dimension [33], we can ensure that, with a confidence level of $1 - \delta$, the number of training patterns in each subset of training patterns obey the following inequalities:
4. $|\hat{X}_i| \geq \max\left(\frac{4}{f(\boldsymbol{w},X)+\epsilon_i} \log \frac{2}{\delta}, \frac{8d^\dagger}{f(\boldsymbol{w},X)+\epsilon_i} \log \frac{13}{f(\boldsymbol{w},X)+\epsilon_i}\right)$
5. $|\hat{X}_{i+1}| \geq \max\left(\frac{4}{f(\boldsymbol{w},X)+\epsilon_{i+1}} \log \frac{2}{\delta}, \frac{8d^\dagger}{f(\boldsymbol{w},X)+\epsilon_{i+1}} \log \frac{13}{f(\boldsymbol{w},X)+\epsilon_{i+1}}\right)$
   where $d\dagger$ is the Vapnik–Chervonenkis dimension for the predictive model. In this case we can consider $d\dagger$ as constant, because the ANN model is the same for every subset of training patterns $\hat{X}_i$.
   From 4 and 5 is easy to see that ϵ_i and ϵ_{i+1} are inversely proportional to $|\hat{X}_i|$ and $|\hat{X}_{i+1}|$ respectively. Hence, considering 3 we can ensure that:
6. $\epsilon_i > \epsilon_{i+1}$ concluding proof. □

Algorithm 1 describes how to use the continuation swarm-based meta-heuristic algorithms for ANN training. The algorithm inputs are a set of training patterns X and a fitness function in the form of $f(\boldsymbol{w}, X)$. In the first step of the algorithm, a restricted sequence of sets of training patterns $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$ is created. The last subset of the sequence, denoted by $\hat{X}_k$, is the entire set of training patterns ($\hat{X}_k = X$) and each subset of training patterns $\hat{X}_i$, $1 \leq i \leq k-1$ is built by randomly removing a $\beta\dagger \cdot p$ number of training patterns from the subset $\hat{X}_{i+1}$ with $\beta\dagger \in (0, 1)$. Hence, the sequence holds that $\hat{X}_1 \subset \hat{X}_2 \subset \ldots \subset \hat{X}_k$.

Algorithm 1 Neural network training by swarm-based meta-heuristics and continuation.

Require: Set of training patterns $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_p}\}$
Require: Fitness function $f(\boldsymbol{w}, X)$, $\boldsymbol{w} \in \mathbb{R}^n$
Ensure: Best particle $\boldsymbol{w}^*$
1: Build a sequence of subsets of training patterns $\hat{X}_1 \subset \hat{X}_2 \subset \ldots \subset \hat{X}_k$
2: Generate initial population $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_r$ with $\boldsymbol{w}_j \in \mathbb{R}^n$
3: **for** i = 1 : k **do**
4: Perform a number of optimization iterations that accounts for η/k fitness function evaluations using $f(\boldsymbol{w}_j, \hat{X}_i)$
5: **end for**
6: **return** best particle $\boldsymbol{w}^*$ in the population

Line of the algorithm introduces an algorithm hyper-parameter η that controls the number of fitness functions evaluations used to optimize the fitness function $f(\boldsymbol{w}_j, \hat{X}_i)$. Note that in Algorithm 1, for each iteration, optimization continues the search using the population from the previous iteration instead of restarting the population randomly. Hence, the name of *continuation*. Finally, the algorithm returns the *best-global* particle of the population.

Despite the difficulty of finding ϵ, we know that the intolerance of a representative sequence of sets training patterns decreases as the sequence increases. We could use this knowledge to increase the accuracy of the decision we make in each optimization steep about the bests solutions.

We could apply Theorem 1 as is $\epsilon = 0$ and still be taking the right choice for a proportion of the cases. As the sequence of sets of training patterns increases, the value of ϵ decreases and the amount of incorrect comparisons between solutions decreases. Hopefully, as we are using meta-heuristics for optimization, the wrong comparisons may be considered as healthy random variations without negative effects in performance as long ϵ is kept within reasonable bounds. Hence, we let the tuning of ϵ as an algorithm hyper-parameter controlled by how many training patterns are in each subset of training patterns $\hat{X}_i$.

Fig. 1 describes graphically Algorithm 1. On the first η/k fitness function evaluations, the optimization algorithm uses the subset $\hat{X}_1$ to calculate the fitness function of each solution. Then, the optimization algorithm uses the solutions found up to this point to continue optimization in the following η/k fitness function evaluations. We expect that irrelevant local irregularities gets removed on the first iterations due to the fitness function simplification effect.

It is worth noticing, that the way we build the restricted sequence of subsets of training patterns is not unique. We can come up with different strategies for sampling and for choosing the proportion of training samples in each subset as long as $\hat{X}_1 \subset \hat{X}_2 \subset \ldots \subset \hat{X}_k$. For example, instead of choosing a fixed proportion $\beta\dagger \cdot p$ of training patterns to be removed from $\hat{X}_{i+1}$, we can sample a number $\beta^\dagger \cdot |\hat{X}_{i+1}|$ of training patterns from $\hat{X}_{i+1}$. Whether one strategy is preferred to another should be considered in future research.

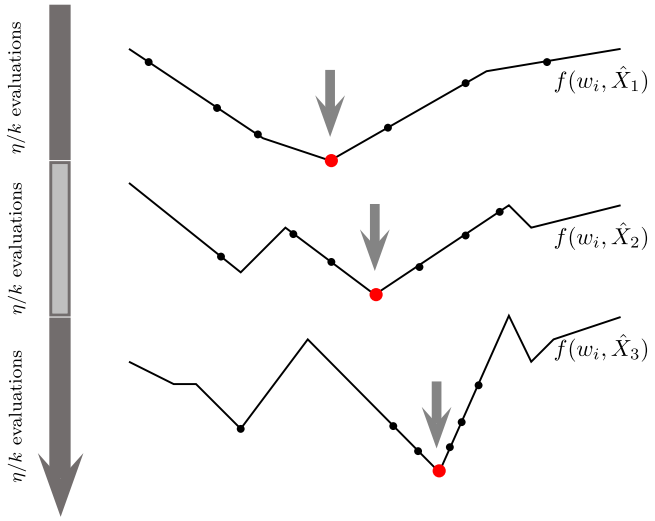

Fig. 1. Graphical representation of optimization by continuation over the curse of iterations.

When considering the execution time of standard meta-heuristic algorithms and its continuation variations, we see that for the standard case the execution time is given by $t_e = p\eta t$ where p is the number of training patterns, η is the number of fitness function evaluations and t is the time of propagating a training pattern in the ANN. For the continuation case the execution time depends of the chosen hyper-parameters $\beta\dagger$ and k:

$$t_c = \sum_{j=0}^{k-1}(p - j\beta^\dagger p)\frac{\eta}{k}t = \left[\frac{1}{k}\sum_{j=0}^{k-1}(1 - j\beta^\dagger)\right]p\eta t$$
$$= \left[\frac{1}{k}\sum_{j=0}^{k}1 - \frac{\beta^\dagger}{k}\sum_{j=0}^{k-1}j\right]p\eta t = \left[\frac{k}{k} - \frac{\beta^\dagger}{k}\frac{(k-1)k}{2}\right]p\eta t$$
$$= \left[1 - \frac{\beta^\dagger(k-1)}{2}\right]p\eta t \quad (5)$$

The execution time for the continuation versions is a fraction of the standard case for valid hyper-parameters values, that is, $\beta\dagger \in (0, 1)$, $k > 1$ and $\beta^\dagger(k-1) < 1$. Consider that when $k = 1$, the continuation algorithm is just the standard meta-heuristic version and $\beta^\dagger(k-1) < 1$ ensures that each subset in the representative sequence contains at least one training pattern.

The previous execution time estimation can be used as a higher bound of efficiency of continuation methods. In particular, in this work we will tune $\beta\dagger$ and k by means of hyper-parameter optimization, which is a non-deterministic process, to prove that a working set of hyper-parameters exists and is able to obtain the same accuracy in a lower execution time.

3.1. Meta-heuristics and hyper-parameters

In this work we will consider three widely studied meta-heuristic algorithms[3]: PSO, FA and CS. We use each algorithm to train ANN. For a comparison with conventional gradient based training algorithms, we also consider SGD algorithm, that in recent studies have shown to be competitive with other methods such as ADAM or RMSProp [34]. SGD have two main hyper-parameters: learning rate (α) and momentum (β).

PSO is a population based meta-heuristic algorithm [35]. Standard PSO have several hyper-parameters such as: global contribution (α), local contribution (β), maximum speed ($v_{\max}$) and minimum speed ($v_{\min}$). In ANN training by PSO, the ith particle in the population represents the ANN parameters vector $\boldsymbol{w}_i$. Let $\boldsymbol{w}^*$ be the position of the particle with the lowest fitness function value in the population and $\boldsymbol{w}_i^*$ be the position where $\boldsymbol{w}_i$ achieved its lowest fitness function value. Each particle $\boldsymbol{w}_i$ moves towards a direction determined by $\boldsymbol{w}^*$, $\boldsymbol{w}_i^*$ and a velocity factor. Particle $\boldsymbol{w}^*$ is also known as *best-global* particle and $\boldsymbol{w}_i^*$ as *best-so-far* particle of $\boldsymbol{w}_i$.

FA is a population based meta-heuristic algorithm [36]. Standard FA have several hyper-parameters such as: initial bright (β), light decay (γ) and random influence (η). In ANN training by FA, the ith firefly in the population represents the ANN parameters vector $\boldsymbol{w}_i$. Each firefly $\boldsymbol{w}_i$ is attracted towards the fireflies with lowest fitness value. The steep size of the movement is based in the bright of the other fireflies and their distance.

CS is a population based meta-heuristic algorithm [37]. Standard CS have several hyper-parameters such as: α related with the problem scale and p_a which is the replacement proportion for worst solutions in each iteration. In ANN by CS, the ith cuckoo represents the ANN parameters vector $\boldsymbol{w}_i$. In each iteration, a random cuckoo is selected from the population and is moved in the parameter space following a LÃ©vy flight. After this, a proportion p_a of the worst cuckoos with high fitness value are replaced.

4. Results and discussion

This section describes the experimental setup and results of the proposed continuation approach to optimize ANNs. The accuracy of training, measured as generalization error, was defined as the MSE on unseen data after 4-folds cross-validation for datasets with less than 1.0E+3 training patterns. For k-fold cross-validation, the Tibshirani–Tibshirani (TT) bias correction for the minimum error rate was considered when performing hyper-parameter optimization [38]. TT is a cheaper alternative to nested-cross-validation. For larger datasets, we considered a training-test-validation (60–20–20%) split.

We selected benchmark datasets considering the number of training patterns from the UCI Machine Learning Repository regression problems [39]. For each dataset, we specify network related hyper-parameters from previous works when available, however, we do not deal with model selection:

1. Concrete Compressive Strength Dataset (CCS). Prediction of concrete compressive strength based in its age and ingredients. The dataset has 1030 training patterns and 9 features. The ANN has two layers of sigmoid units with 17 neurons in the hidden layer and 1 output neuron according to [40].
2. Wine Quality Dataset (WQ). Prediction of wine quality based in a series of physics and chemistry features of wine. The dataset has 4898 training patterns and 12 features. The ANN has two layers of sigmoid units with 14 neurons in the hidden layer and 1 output neuron according to [41].
3. Combined Cycle Power Plant Dataset (CCPP). Prediction of the amount of electrical energy that a specific power plant is able to produce each hour. The dataset has 9568 training patterns and 4 features. The ANN has two layers of sigmoid units with 14 neurons in the hidden layer and 1 output neuron according to [42].
4. Bike Sharing Dataset (BKS). Prediction of the number of bikes used in a bike rental service. The dataset has 17,389 training patterns and 20 attributes. The ANN has two layers of sigmoid units with 45 neurons in the hidden layer and 1 output neuron.
5. Pollution Dataset (PLN). Prediction of the amount of pollution in Beijing city. The dataset has 43,824 training patterns and 16

[3] Source code at: https://github.com/jairodelgado/dnn_opt.

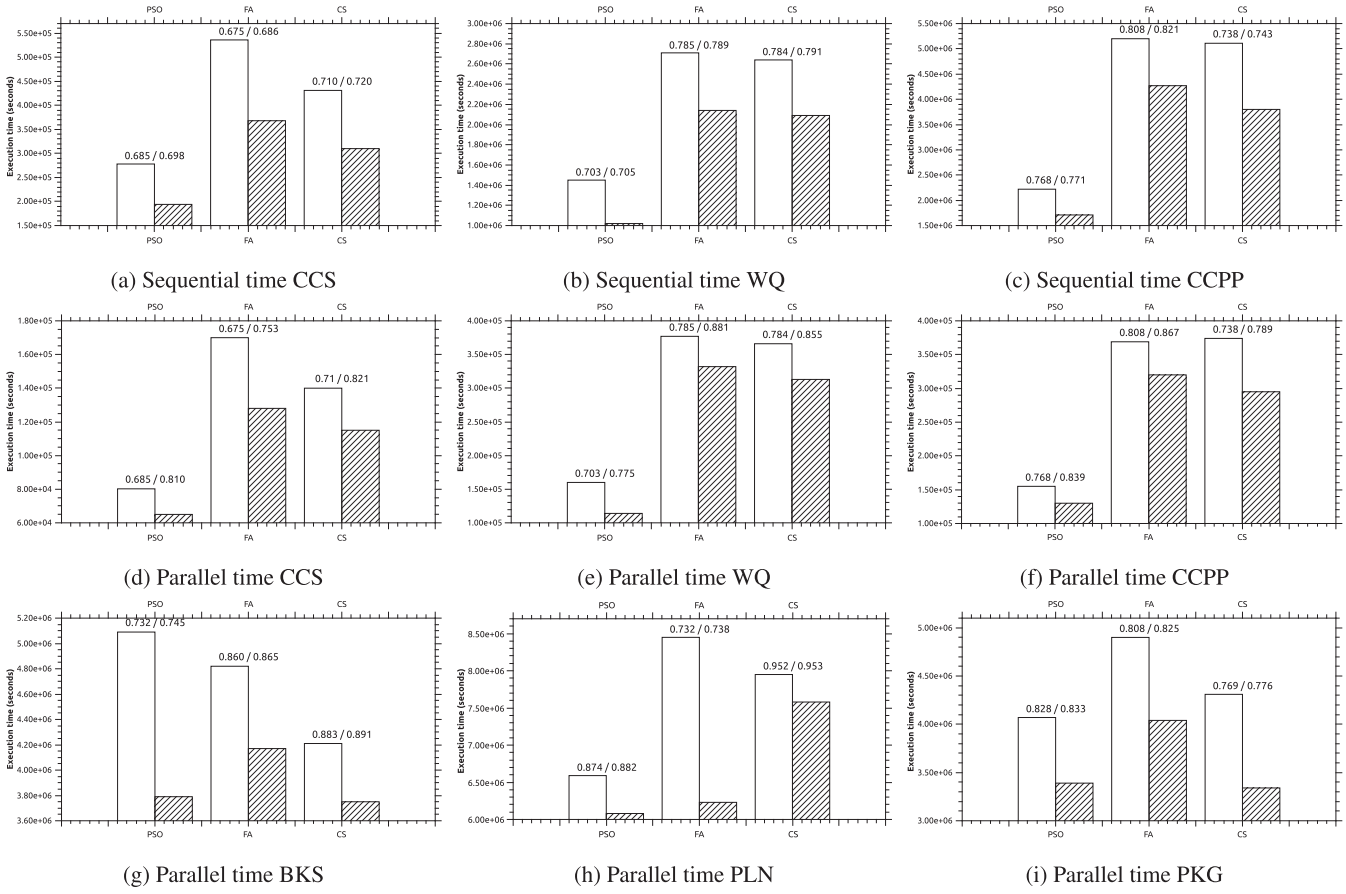

Fig. 2. Execution time of standard meta-heuristic algorithms and continuation meta-heuristic algorithms for sequential and parallel implementations.

attributes. The ANN has two layers of sigmoid units with 45 neurons in the hidden layer and 1 output neuron.

6. Car Park Occupancy Dataset (PKG). Prediction of the number of spots used in a parking service. The dataset has 35717 training patterns and 8 attributes. The ANN has two layers of sigmoid units with 45 neurons in the hidden layer and 1 output neuron.

Table 1 presents a comprehensive summary of selected hyper-parameters for each dataset. Such hyper-parameters can be used to reproduce experimental results. Three widely studied meta-heuristic algorithms were used to perform optimization: Particle Swarm Optimization (PSO), Firefly Algorithm (FA) and Cuckoo Search (CS). In addition, we compare the results of these meta-heuristic algorithms and its continuation versions with SGD algorithm. We configure hyper-parameters related to training algorithms using SMAC [43], an automatic hyper-parameter optimization algorithm. For small datasets (CCS, WQ and CCPP) we use a population size of 40 and for large datasets (BKS, PLN and PKG) we use a population size of 200.

To summarize the experimental methodology followed in this paper, in a first stage we optimize hyper-parameter by means of SMAC algorithm for each dataset and meta-heuristic algorithm. To estimate generalization error, we use k-fold cross-validation with TT for small datasets (CCS, WQ and CCPP) and for large datasets (BKS, PLN and PKG) we use a regular training-test-validation split. In a second stage, we calculate the execution time of training with the selected hyper-parameters from the first stage and using all training patterns in the dataset.

Table 2 shows the generalization error and training error obtained by each training algorithm. Results are presented in the form of mean (standard deviation) of 50 measurements. In bold typeface we show the best results for each dataset. For each meta-heuristic algorithm we used a fixed number of 4.0E+4 fitness function evaluations as stopping condition. For SGD we used a number of 1.0E+4 epochs with a mini-batch size of 40 training patterns.

In order to find statistical significant differences between standard and continuation meta-heuristic algorithms we verified normality hypothesis with Anderson-Darling goodness of fit test. T-Student test, presented in Table 3, was performed when considering generalization error. In bold typeface we show statistically significant differences for a 95% of confidence.

As can be seen, the generalization error of standard meta-heuristics and continuation based meta-heuristics do not show statistical significant differences for most of the datasets. The algorithm SGD obtained higher generalization error than the meta-heuristic algorithms. However, for larger datasets (BKS, PLN, PKG) only PSO outperforms SGD. This may show that FA and CS present issues when dealing with a large number of optimization parameters. Note the reduced standard deviation of generalization error for continuation meta-heuristic algorithms respect standard meta-heuristic algorithms. This suggest a more stable optimization process.

Table 4 shows the difference between training error and generalization error. We observe a small reduction of the gap between generalization error and training error of the continuation meta-heuristic algorithms respect standard meta-heuristic algorithms. Further analysis should be conducted to study the possible regularization effect of optimization by continuation.

Fig. 2 presents the execution time average after 50 measurements for the benchmark datasets. Empty bars correspond to standard meta-heuristic algorithms and filled bars correspond to

Table 1
Hyper-parameter configuration for SGD algorithm, standard meta-heuristic algorithms and continuation meta-heuristic algorithms.

	SGD			PSO							FA						CS				
	α	β	TT	α	β	*max*	*min*	β†	k	TT	β	γ	η	β†	k	TT	α	p_a	β†	k	TT
	Standard meta-heuristic algorithms																				
CSS	0.128	0.162	1.37E-4	0.945	0.372	1.252	0.763	–	–	1.41E-3	0.931	2.54E-4	0.601	–	–	3.12E-4	0.176	0.512	-	-	2.12E-4
WQ	0.782	0.061	4.65E-4	0.942	0.370	1.254	0.758	–	–	2.48E-4	0.939	2.56E-4	0.594	–	–	2.54E-4	0.682	0.766	–	-	1.79E-4
CCPP	0.037	0.025	6.41E-5	0.931	0.359	1.248	0.763	–	–	1.81E-5	0.911	9.83E-4	0.258	–	–	5.01E-7	0.987	0.973	–	-	6.84E-7
BKS	0.032	0.047	–	0.917	0.794	0.740	0.888	–	–	–	0.934	0.573	0.145	–	–	–	0.869	0.686	–	-	-
PLN	0.571	0.067	–	0.939	0.902	0.994	0.600	–	–	–	0.156	0.080	0.568	–	–	–	0.375	0.992	–	–	–
PKG	0.041	0.371	–	0.218	0.897	0.723	0.605	–	–	–	0.844	0.945	0.478	–	–	–	0.121	0.079	–	–	–
	Continuation meta-heuristic algorithms																				
CSS	-	-	–	0.911	0.522	1.063	0.802	0.217	4	4.03E-4	0.890	8.33E-4	0.264	0.130	6	4.50E-5	0.063	0.727	0.145	5	8.31E-5
WQ	-	-	–	0.760	0.883	0.940	0.784	0.198	4	1.12E-4	0.752	6.91E-4	0.067	0.431	2	1.43E-4	0.221	0.974	0.433	2	6.70E-4
CCPP	-	-	–	0.648	0.718	1.073	0.827	0.464	2	1.05E-5	0.977	1.13E-2	0.081	0.048	9	3.30E-8	0.063	0.727	0.075	8	9.20E-8
BKS	-	-	–	0.646	0.911	0.623	0.820	0.067	9	–	0.022	0.552	0.591	0.056	6	–	0.552	0.376	0.039	7	-
PLN	-	-	–	0.724	0.814	0.632	0.922	0.063	5	–	0.072	0.723	0.790	0.067	9	–	0.303	0.443	0.016	7	-
PKG	-	-	–	0.292	0.966	0.722	0.755	0.043	9	–	0.385	0.860	0.300	0.055	8	–	0.983	0.745	0.066	8	-

Table 2
Accuracy of ANN training considering SGD algorithm, standard meta-heuristic algorithms and continuation meta-heuristic algorithms.

	Gradient	Standard			Continuation		
	SGD	SPSO	SFA	SCS	CPSO	CFA	CCS
				Generalization error			
CCS	4.34E-2 (1.28E-4)	1.00E-2 (9.99E-4)	9.62E-3 (5.25E-4)	1.20E-2 (1.69E-3)	**8.50E-3 (7.95E-4)**	9.79E-3 (3.40E-4)	1.16E-2 (1.53E-3)
WQ	2.24E-2 (2.41E-4)	1.61E-2 (2.76E-4)	1.68E-2 (1.40E-4)	1.67E-2 (1.43E-4)	**1.60E-2 (2.16E-4)**	1.69E-2 (1.23E-4)	1.65E-2 (2.10E-4)
CCPP	5.12E-2 (1.64E-5)	**3.13E-3 (3.79E-5)**	3.21E-3 (2.30E-5)	3.23E-3 (2.87E-5)	**3.13E-3 (3.12E-5)**	3.21E-3 (2.33E-5)	3.23E-3 (2.66E-5)
BKS	1.49E-2 (3.44E-4)	3.09E-3 (2.85E-4)	3.82E-2 (6.49E-3)	2.40E-2 (2.16E-3)	**2.65E-3 (1.18E-4)**	3.80E-2 (6.15E-3)	2.38E-2 (1.75E-3)
PLN	8.68E-3 (2.66E-4)	**7.82E-3 (1.80E-4)**	1.56E-2 (1.27E-3)	1.24E-2 (8.21E-4)	7.97E-3 (1.60E-4)	1.61E-2 (1.15E-3)	1.21E-2 (4.51E-4)
PKG	8.37E-3 (2.13E-4)	**6.52E-3 (6.61E-5)**	2.37E-2 (2.35E-3)	1.58E-2 (1.43E-3)	**6.52E-3 (5.44E-5)**	2.23E-2 (1.90E-3)	1.60E-2 (8.99E-4)
				Training error			
CCS	4.33E-2 (2.02E-5)	6.98E-3 (7.36E-4)	8.61E-3 (4.50E-4)	1.11E-2 (1.52E-3)	6.48E-3 (5.71E-4)	8.51E-3 (2.97E-4)	1.10E-2 (1.51E-3)
WQ	2.19E-2 (2.13E-4)	1.51E-2 (2.05E-4)	1.63E-2 (1.03E-4)	1.63E-2 (1.05E-4)	1.52E-2 (1.54E-4)	1.66E-2 (1.24E-4)	1.60E-2 (1.55E-4)
CCPP	5.11E-2 (5.07E-6)	3.09E-3 (3.67E-5)	3.20E-3 (2.21E-5)	3.22E-3 (2.61E-5)	3.11E-3 (3.17E-5)	3.20E-3 (2.36E-5)	3.01E-3 (2.96E-5)
BKS	1.36E-2 (4.08E-5)	2.74E-3 (4.01E-4)	3.42E-2 (1.10E-3)	2.20E-2 (1.09E-3)	2.42E-3 (2.32E-4)	3.75E-2 (9.69E-4)	2.24E-2 (7.98E-4)
PLN	8.67E-3 (3.13E-6)	6.91E-3 (5.60E-4)	1.23E-2 (8.92E-4)	1.15E-2 (7.40E-4)	6.54E-4 (1.39E-4)	1.48E-2 (1.12E-3)	1.18E-2 (5.74E-4)
PKG	8.09E-3 (5.68E-7)	6.35E-3 (9.62E-5)	2.17E-2 (8.64E-4)	1.38E-2 (2.17E-3)	6.47E-3 (7.90E-5)	2.15E-2 (4.73E-4)	1.37E-2 (5.93E-4)

Table 3
T-Student test p-value and t-value of comparing standard and continuation meta-heuristic algorithms generalization error.

	CCS			WQ			CCPP			BKS			PLN			PKG		
	PSO	FA	CS	PSO	FA	CS	PSO	FA	CS	PSO	FA	CS	PSO	FA	CS	PSO	FA	CS
p-value	**0.002**	0.06	0.09	0.69	0.77	0.96	0.82	0.95	0.61	0.09	0.06	0.09	0.14	0.23	0.47	0.84	0.91	0.14
t-value	**8.41**	1.89	1.68	2.49	4.82	5.41	0.21	0.06	0.50	8.41	1.89	1.68	2.74	3.47	6.28	0.34	0.09	0.88

Table 4
Difference between training error and generalization error for each optimization algorithm and dataset.

	Gradient	Standard			Continuation		
	SGD	SPSO	SFA	SCS	CPSO	CFA	CCS
CCS	1.00E-04	3.02E-03	1.01E-03	9.00E-04	2.02E-03	1.28E-03	6.00E-04
WQ	5.00E-04	1.00E-03	5.00E-04	4.00E-04	8.00E-04	3.00E-04	5.00E-04
CCPP	1.00E-04	4.00E-05	1.00E-05	1.00E-05	2.00E-05	1.00E-05	2.20E-03
BKS	1.30E-03	3.50E-04	4.00E-03	2.00E-03	2.30E-04	5.00E-04	1.40E-03
PLN	1.00E-05	9.10E-04	3.30E-03	9.00E-04	3.70E-03	1.30E-03	3.00E-04
PKS	2.80E-04	1.70E-04	2.00E-03	2.00E-03	5.00E-05	8.00E-04	2.30E-03

continuation meta-heuristic algorithms. The label in the top of the bars represent the expected theoretical improvement / experimental execution time improvement. We refer to theoretical improvement to the expected execution time reduction according to Eq. (5), $1-\beta^{\dagger}(k-1)/2$. Experimental execution time improvement is the actual execution time of the continuation meta-heuristic algorithm over the execution time of the standard meta-heuristic algorithm in a given implementation and hardware platform as described below.

Execution time was measured in seconds in a personal computer with a Core-i3 microprocessor at 2.3 GHz with 4Gb of RAM using a sequential implementation and in an NVIDIA GeForce 920M GPU with 320 stream processors and 2Gb of DRAM. For datasets BKS, PLN and PKG we only provide GPU execution time.

In general, the continuation approach for optimization represents a significant diminution in execution time for all meta-heuristic algorithm and benchmark dataset. Optimization time was reduced depending on the chosen continuation hyper-parameters for sequential and parallel implementations. We see that Eq. (5) can predict the improvement in execution time of continuation respect standard meta-heuristic algorithms. However, depending on the specific implementation, the experimental execution time may be higher than the expected execution time. For example, in the case parallel GPU implementations we see that for small datasets (CCS, WQ and CCPP) the experimental execution time reduction is smaller than the expected when compared with larger datasets. This is because of the vectorial execution model of GPUs.

5. Conclusions and recommendations

In this paper we presented an approach for ANN training based on optimization by continuation and swarm-based algorithms. A considerable diminution in execution time of optimization is observed when using the continuation approach for sequential and parallel GPU implementations. Further analysis should be conducted to investigate alternative sampling strategies to build restricted sequences of subsets of training patterns and the possible regularization effect of optimization by continuation.

Conflict of interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

References

[1] F.W. Glover, G.A. Kochenberger, Handbook of Metaheuristics, 57, Springer Science & Business Media, 2006.
[2] K.-L. Du, M. Swamy, et al., Search and Optimization by Metaheuristics, Birkhaüser, July, 2016.

[3] R. Livni, S. Shalev-Shwartz, O. Shamir, On the computational efficiency of training neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 855–863.
[4] M. Janzamin, H. Sedghi, A. Anandkumar, Beating the perils of non-convexity: guaranteed training of neural networks using tensor methods, CoRR (2015). abs/1506.08473
[5] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[6] Y.N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 2933–2941.
[7] M. Yu, P. Yang, S. Wei, Railway obstacle detection algorithm using neural network, in: AIP Conference Proceedings, 1967, AIP Publishing, 2018, p. 040017.
[8] S. Owaki, Y. Fukumoto, T. Sakamoto, N. Yamamoto, M. Nakamura, Experimental demonstration of SPM compensation based on digital signal processing using a three-layer neural-network for 40-gbit/s optical 16qam signal, IEICE Commun. Express 7 (1) (2018) 13–18.
[9] K. Li, S. Mao, X. Li, Z. Wu, H. Meng, Automatic lexical stress and pitch accent detection for l2 english speech using multi-distribution deep neural networks, Speech Commun. 96 (2018) 28–36.
[10] A. Griewank, et al., On automatic differentiation, Math. Program. 6 (6) (1989) 83–107.
[11] S. Chatterjee, N. Dey, A.S. Ashour, C.V.A. Drugarin, Electrical energy output prediction using cuckoo search based artificial neural network, in: Smart Trends in Systems, Security and Sustainability, Springer, 2018, pp. 277–285.
[12] M. Sreeshakthy, J. Preethi, Classification of human emotion from deap eeg signal using hybrid improved neural networks with cuckoo search, BRAIN 6 (3–4) (2016) 60–73.
[13] M. Hashem, A.S. Hassanein, Jaw fracture classification using meta heuristic firefly algorithm with multi-layered associative neural networks, Cluster Comput. (2018) 1–8.
[14] J. Nayak, B. Naik, H. Behera, A novel nature inspired firefly algorithm with higher order neural network: performance analysis, Eng. Sci. Technol. Int. J. (2015).
[15] H.M. Ahmed, B.A. Youssef, A.S. Elkorany, A.A. Saleeb, F.A. El-Samie, Hybrid gray wolf optimizer–artificial neural network classification approach for magnetic resonance brain images, Appl. Opt. 57 (7) (2018) B25–B31.
[16] N.M. Nawi, M. Rehman, A. Khan, Ws-bp: An efficient wolf search based backpropagation algorithm, in: International Conference on Mathematics, Engineering and Industrial Applications, 1660, AIP Publishing, 2015, p. 050027.
[17] P. Chang, J. Yang, J. Yang, T. You, Respiratory signals prediction based on particle swarm optimization and back propagation neural networks, in: AIP Conference Proceedings, 1834, AIP Publishing, 2017, p. 040049.
[18] W. Chen, X.A. Wang, W. Zhang, C. Xu, Phishing detection research based on pso-bp neural network, in: International Conference on Emerging Internetworking, Data & Web Technologies, Springer, 2018, pp. 990–998.
[19] T. Chilimbi, Y. Suzue, J. Apacible, K. Kalyanaraman, Project adam: Building an efficient and scalable deep learning training system, in: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 2014, pp. 571–582.
[20] H. Mobahi, J.W. Fisher, A theoretical analysis of optimization by gaussian continuation., in: AAAI, 2015, pp. 1205–1211.
[21] E.L. Allgower, K. Georg, Numerical Continuation Methods: An Introduction, 13, Springer Science & Business Media, 2012.
[22] T. Ash, Dynamic node creation in backpropagation networks, Connect. Sci. 1 (4) (1989) 365–375.
[23] K. Parsopoulos, V. Plagianakos, G. Magoulas, M. Vrahatis, Stretching technique for obtaining global minimizers through particle swarm optimization, in: Proceedings of the Particle Swarm Optimization Workshop, 29, Indianapolis, USA, 2001.
[24] V. Losing, B. Hammer, H. Wersing, Incremental on-line learning: a review and comparison of state of the art algorithms, Neurocomputing 275 (2018) 1261–1274.
[25] A. Gepperth, B. Hammer, Incremental learning algorithms and applications (2016).
[26] D. Mellado, C. Saavedra, S. Chabert, R. Salas, Pseudorehearsal approach for incremental learning of deep convolutional neural networks (2017) 118–126.
[27] K. Shmelkov, C. Schmid, K. Alahari, Incremental learning of object detectors without catastrophic forgetting, in: 2017 IEEE International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 3420–3429.
[28] M. Pedergnana, S.G. García, et al., Smart sampling and incremental function learning for very large high dimensional data, Neural Netw. 78 (2016) 75–87.
[29] C.A. Tovey, Simulated simulated annealing, Am. J. Math. Manag. Sci. 8 (3–4) (1988) 389–407.
[30] Z.-D. Jian, T.-S. Hsu, D.-W. Wang, Searching vaccination strategy with surrogate-assisted evolutionary computing, in: Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), 2016 6th International Conference on, IEEE, 2016, pp. 1–8.
[31] Ž. Lukšič, J. Tanevski, S. Džeroski, L. Todorovski, General meta-model framework for surrogate-based numerical optimization, in: International Conference on Discovery Science, Springer, 2017, pp. 51–66.
[32] P. Hansen, N. Mladenović, R. Todosijević, S. Hanafi, Variable neighborhood search: basics and variants, EURO J. Comput. Optim. 5 (3) (2017) 423–454.
[33] A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth, Learnability and the vapnik-chervonenkis dimension, J. ACM 36 (4) (1989) 929–965.
[34] A. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht, The marginal value of adaptive gradient methods in machine learning, in: NIPS, 18, 2017, pp. 115–122.
[35] M.R. Bonyadi, Z. Michalewicz, Particle swarm optimization for single objective continuous space problems: a review, 2017.
[36] X.-S. Yang, X. He, Firefly algorithm: recent advances and applications, Int. J. Swarm Intell. 1 (1) (2013) 36–50.
[37] M. Shehab, A.T. Khader, M.A. Al-Betar, A survey on applications and variants of the cuckoo search algorithm, Appl. Soft Comput. 61 (2017) 1041–1059.
[38] R.J. Tibshirani, R. Tibshirani, A bias correction for the minimum error rate in cross-validation, Ann. Appl. Stat. (2009) 822–829.
[39] M. Lichman, UCI machine learning repository, 2013.
[40] D. Neeraja, G. Swaroop, Prediction of compressive strength of concrete using artificial neural networks, Res. J. Pharm. Technol. 10 (1) (2017) 35–40.
[41] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, Modeling wine preferences by data mining from physicochemical properties, Decis. Support Syst. 47 (4) (2009) 547–553.
[42] M. Rashid, K. Kamal, T. Zafar, Z. Sheikh, A. Shah, S. Mathavan, Energy prediction of a combined cycle power plant using a particle swarm optimization trained feedforward neural network, in: Mechanical Engineering, Automation and Control Systems (MEACS), 2015 International Conference on, IEEE, 2015, pp. 1–5.
[43] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration., LION 5 (2011) 507–523.