Bayesian neural architecture search using a training-free performance metric

Camero Unzueta, A.; Wang, H.; Alba, E.; Bäck, T.H.W.

Applied Soft Computing 106 (2021) 107356

Contents lists available at ScienceDirect

Applied Soft Computing

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

Bayesian neural architecture search using a training-free performance metric

Andrés Camero [a,b,*], Hao Wang [c], Enrique Alba [b], Thomas Bäck [c]

[a] German Aerospace Center (DLR), Remote Sensing Technology Institute (IMF), Germany
[b] Universidad de Málaga, ITIS Software, Spain
[c] Leiden University, LIACS, The Netherlands

ARTICLE INFO

Article history:
Received 3 September 2020
Received in revised form 10 March 2021
Accepted 13 March 2021
Available online 29 March 2021

Keywords:
Bayesian optimization
Recurrent neural network
Neural architecture search
Architecture optimization

ABSTRACT

Recurrent neural networks (RNNs) are a powerful approach for time series prediction. However, their performance is strongly affected by their architecture and hyperparameter settings. The architecture optimization of RNNs is a time-consuming task, where the search space is typically a mixture of real, integer and categorical values. To allow for shrinking and expanding the size of the network, the representation of architectures often has a variable length. In this paper, we propose to tackle the architecture optimization problem with a variant of the Bayesian Optimization (BO) algorithm. To reduce the evaluation time of candidate architectures the Mean Absolute Error Random Sampling (MRS), a training-free method to estimate the network performance, is adopted as the objective function for BO. Also, we propose three fixed-length encoding schemes to cope with the variable-length architecture representation. The result is a new perspective on accurate and efficient design of RNNs, that we validate on three problems. Our findings show that (1) the BO algorithm can explore different network architectures using the proposed encoding schemes and successfully designs well-performing architectures, and (2) the optimization time is significantly reduced by using MRS, without compromising the performance as compared to the architectures obtained from the actual training procedure.

1. Introduction

With the advent of deep learning, deep neural networks (DNNs) have gained popularity, and they have been applied to a wide variety of problems [1,2]. When it comes to sequence modeling and prediction, Recurrent Neural Networks (RNNs) have proved to be the most suitable ones [2]. Essentially, RNNs are feedforward networks with feedback connections. This feature allows them to capture long-term dependencies among the input variables. Despite their good performance, they are very sensitive to their *hyperparameter* configuration and hard to train [1,3–5].

Finding an appropriate hyperparameter setting has always been a difficult task. The conventional approach to tackle this problem is to do a trial/error exploration based on expert knowledge. In other words, a human expert defines an architecture, sets up a training method (usually a gradient descent-based algorithm), and performs the training of the network until some criterion is met. Lately, automatic methods based on optimization algorithms, e.g., grid search, evolutionary algorithms or Bayesian optimization (BO), have been proposed to *replace* the human expert. However, due to the immense size and complexity of the search space, and the high computational cost of training a DNN, hyperparameter optimization still poses an open problem [1,4].

Different approaches have been proposed for improving the performance of hyperparameter optimization, ranging from evolutionary approaches (a.k.a. neuroevolution) [4], to techniques to speed up the evaluation of a DNN [6,7]. Among these approaches, the *Mean Absolute Error Random Sampling* (MRS) [6] poses a promising "low-cost, training-free, rule of thumb" alternative to evaluate the performance of an RNN, which drastically reduces the evaluation time.

In this study, we propose to tackle the architecture optimization problem with a hybrid approach. Specifically, we combine BO [8,9] for optimizing the architecture, MRS [6] for evaluating the performance of candidate architectures, and ADAM [10] (a gradient descent-based algorithm) truncated through time for training the *final* architecture on a given problem. We benchmark our proposal on three problems (the sine wave, the filling level of 217 recycling bins in a metropolitan area, and the load demand forecast of an electricity company in Slovakia) and compare our results against the state-of-the-art.

[*] Corresponding author at: German Aerospace Center (DLR), Remote Sensing Technology Institute (IMF), Germany.
E-mail addresses: andres.camerounzueta@dlr.de (A. Camero), h.wang@liacs.leidenuniv.nl (H. Wang), eat@lcc.uma.es (E. Alba), t.h.w.baeck@liacs.leidenuniv.nl (T. Bäck).

https://doi.org/10.1016/j.asoc.2021.107356
1568-4946/© 2021 Elsevier B.V. All rights reserved.

Therefore, the main contributions of this study are:

- We define a method to optimize the architecture of an RNN based on BO and MRS that significantly reduces the time without compromising the performance (error),
- We introduce multiple alternatives to cope with the variable-length solution problem. Specifically, we study three encoding schemes and two penalty approaches (i.e., the infeasible representation and the constraint handling), and
- We propose a strategy to improve the performance of the surrogate model of BO for variable-length solutions based on the augmentation of the initial set of solutions, i.e., the *warm-start*.

The remainder of this article is organized as follows: Section 2 briefly reviews some of the most relevant works related to our proposal. Section 3 introduces our proposed approach. Section 4 presents the experimental study, and Section 5 provides conclusions and future work.

2. Related work

In this Section, we summarize some of the most relevant works related to our proposal. First, we introduce the architecture optimization problem and some interesting proposals to tackle it in Section 2.1. Second, we present the *neuroevolution*, a research line for handling the problem (Section 2.2). After briefly reviewing the Mean Absolute Error Random Sampling (MRS) method in Section 2.3 we finally introduce Bayesian Optimization in Section 2.4.

2.1. Architecture optimization

The existing literature teaches us on the importance of optimizing the architecture of a deep neural network on a particular problem, including, for example, the type of activation functions, the number of hidden layers, and the number of units for each layer [11–13]. For DNNs, the architecture optimization task is usually faced by either manual exploration of the search space (that is usually guided by expert knowledge) or by automatic methods based on optimization algorithms, e.g., grid search, evolutionary algorithms or Bayesian optimization [4].

The challenges here are three-fold: firstly, the search space is typically huge due to the fact that the number of the parameters increases in proportion to the number of layers. Secondly, the search space is usually a mixture of real (e.g., the weights), integer (e.g., the number of units in each layer) and categorical (e.g., the type of activation functions) values, resulting in a demanding optimization task: different types of parameters naturally require different approaches for handling them in optimization. Last, the architecture optimization falls into the family of expensive optimization problems as function evaluations in this case are highly time consuming (which is affected both by the size of training data and the depth of the architecture). In this paper, we shall denote the search space of architecture optimization as $\mathcal{H}$. The specification of $\mathcal{H}$ depends on the choice of encoding schemes of the architecture (see Section 3.1).

To tackle the mentioned issues, many alternatives have been explored, ranging from reducing the evaluation time of a configuration (e.g., early stopping criteria based on the learning curve [7] or MRS [6]) to *evolving* the architecture of the network (neuroevolution).

On the other hand, when it comes to RNN optimization, there are two particular issues: the exploding and the vanishing gradient [3]. Many alternatives have been proposed to tackle with this problems [5]. One of the most popular ones is the Long Short Term Memory (LSTM) cell [14]. However, in spite of its ability to effectively deal with these issues, the problem still remains open, because the learning process is also affected by the weight initialization strategy [15] and the algorithm parameters [1].

2.2. Neuroevolution

Neuroevolutionary approaches typically represent the DNN architecture as solution candidates in specifically designed variants of state-of-the-art evolutionary algorithms. For instance, genetic algorithms (GA) have been applied to evolve increasingly complex neural network topologies and the weights simultaneously, in the so-called NeuroEvolution of Augmenting Topologies (NEAT) method [16,17]. However, NEAT has some limitations when it comes to evolving RNNs [18], e.g., the fitness landscape is deceptive and a large number of parameters have to be optimized. For RNNs, NEAT-LSTM [19] and CoDeepNeat [20] extend NEAT to mitigate its limitations when evolving the topology and weights of the network. Besides NEAT, there are several evolutionary-based approaches to evolve an RNN, such as EXALT [21], EXAMM [22], or a method using ant colony optimization (ACO) to improve LSTM RNNs by refining their cellular structures [23].

A recent work [24] suggested to address the issue of huge training costs when evolving the architecture. In that research, the objective function, that is usually evaluated by training the candidate network on the full data set evolved by a complete training of the candidate network, instead it is approximated by the so-called MAE random sampling (MRS) method, in which no actual training is required. In this manner, the time required for a function evaluation is drastically reduced in the architecture optimization process.

2.3. Mean Absolute Error Random Sampling

MAE Random Sampling is an approach to evaluate the expected error performance of a given architecture. First, the weights of the network are randomly initialized. Second, the error is calculated (i.e., the real and expected output are compared). This two-step process is repeated, and the errors are accumulated. Then, a probabilistic density function (e.g., a truncated normal distribution) is fitted to the error values. Finally, the probability of finding a set of weights whose error is below a user-defined *threshold* is estimated. In other words, by using a random sampling of the output (error), we are estimating how *easy* (i.e., a high probability) it would be to find a *good* (i.e., small error) set of weights.

Given a training data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_i^N$, $\mathbf{x}_i \in \mathbb{R}^n$, for a given network architecture $\mathbf{h} \in \mathcal{H}$ and Q i.i.d. random weight matrices $\{\mathbf{W}_i\}_{i=1}^Q$, $\mathbf{W}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the *Mean Absolute Error* (MAE) of this RNN is denoted as $\mathcal{E} = \{\text{MAE}(\mathcal{D}, \mathbf{h}, t, \mathbf{W}_i)\}_{i=1}^Q$, where t is the number of time steps in the past used for the prediction. Let μ and σ denote the sample mean and standard deviation of the error sample $\mathcal{E}$. Then the so-called *Mean Absolute Error Random Sampling* (MRS) measure is defined as the empirical probability of obtaining a better error rate than a user-specified threshold p_m:

$$\text{MRS}(\mathcal{D}, \mathbf{h}, t, p_\text{m}, Q) = \frac{\Phi\left(\frac{p_\text{m}-\mu}{\sigma}\right) - \Phi\left(-\frac{\mu}{\sigma}\right)}{1 - \Phi\left(-\frac{\mu}{\sigma}\right)}, \tag{1}$$

where Φ stands for the cumulative distribution function (CDF) of the standard normal distribution. The MRS value is calculated from a truncated normal distribution (on the interval $[0, \infty)$), whose location and scale parameters are set to the sample mean μ and standard deviation σ, respectively. Throughout this paper, we shall set p_m to 1%. Intuitively, the higher MRS value a network architecture that yields a higher MRS value would be more likely to possess a much smaller (hence better) MAE rate after the back-propagation training. Hence, it seems promising to use MRS as a training-free estimation for the performance of neural networks.

In this paper, we shall adopt MRS as the objective function (that is subject to maximization) for the architecture optimization. For a detailed discussion of MRS, please refer to [6].

2.4. Bayesian optimization

The so-called *Bayesian Optimization* (BO) (a.k.a. Efficient Global Optimization) [8,9] algorithm has been applied extensively for automated algorithm configuration tasks [25–27]. Bayesian optimization is a sequential global optimization strategy that does not require the derivatives of the objective function and is designed to tackle expensive global optimization problems. Given a real-valued *maximization problem* $f\colon \mathcal{H} \to \mathbb{R}$ (e.g., $f = \textsc{mrs}$ in the following), BO employs a surrogate model, e.g., Gaussian process regression (GPR) or random forests (RF), to approximate the landscape of the objective function, which is trained on an initial data set (X, Y). Here, $X \subset \mathcal{H}$ is typically sampled in the search space $\mathcal{H}$ using the Latin Hypercube Sampling (LHS) method and $Y = \{f(\mathbf{h})\colon \mathbf{h} \in X\}$ is the set of function values of points in X. Essentially, the prediction from surrogate models and the estimated prediction uncertainty are considered simultaneously to propose new candidate solutions for the evaluation. Loosely speaking, the model prediction and its uncertainty are taken as input to the so-called *acquisition function* (or *infill criterion*), which can be interpreted as the utility of unseen solutions and hence is subject to maximization when proposing new candidate solutions. An example of commonly used acquisition functions is the Expected Improvement (EI) [9]. Given the predictor $m\colon \mathcal{H} \to \mathbb{R}$, the uncertainty of predictions $s(\mathbf{h}) := \mathbb{E}\{(m(\mathbf{h}) - f(\mathbf{h}))^2\}$ of the surrogate model and the current best function value $y_{\max} = \max\{Y\}$, the EI criterion can be expressed for an unknown point $\mathbf{h} \in \mathcal{H}$:

$$\mathrm{EI}(\mathbf{h}) = I(\mathbf{h})\Phi\left(\frac{I(\mathbf{h})}{s(\mathbf{h})}\right) + s(\mathbf{h})\phi\left(\frac{I(\mathbf{h})}{s(\mathbf{h})}\right), \tag{2}$$

where $I(\mathbf{h}) = m(\mathbf{h}) - y_{\max}$ and where ϕ stands for the probability density function (PDF) of the standard normal distribution. Note that the new candidate solution is generated by maximizing the EI criterion, namely

$$\mathbf{h}^* = \arg\max_{\mathbf{h}\in\mathcal{H}} \mathrm{EI}(\mathbf{h}). \tag{3}$$

After evaluating the new candidate solution $\mathbf{h}^*$, $\mathbf{h}^*$ and its objective function value are included in the data set (X, Y) and the surrogate model will be re-trained. Please, see [28] for an overview of the acquisition functions.

Despite being a proven technique for automated algorithm configuration tasks [25–27], the state-of-the-art of BO does not reconcile well with variable-length solution problems [29,30]. Therefore, in this study we propose multiple strategies to cope with variable-length solutions (inherent to the architecture search problem).

2.5. Our contribution

Herein, we briefly summarize the novelty of the architecture search described in the following sections and compare those to the state-of-the-art works reviewed in this section.

- We propose to use the Mean Absolute Error Random Sampling (MRS) procedure as the objective function for the architecture search, which is relatively much inexpensive compared to full training of the same architecture on the same data. In contrast to employing full training, e.g., [19], our approach could allow for more iterations of the Bayesian optimization algorithm.
- We designed three different encoding schemes that turn the neural architecture search that is inherently a variable-dimensional problem(for instance, the NEAT [16] approach operates on the network topology directly) into an optimization problem with fixed dimensions, hence facilitating the application of surrogate modeling and Bayesian optimization accordingly.
- We contemplated making the Bayesian optimization more efficient and effective by imposing a penalty on the infeasible solutions or warm-starting the search process with infeasible solutions.

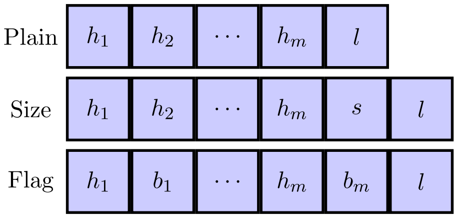


Fig. 1. Illustrations of the proposed encoding schemes.

3. The proposed approach

In this paper, it is proposed to optimize the architecture of an RNN by a combination of Bayesian optimization (BO) and Mean Absolute Error Random Sample (MRS) to reduce the running time of the architecture search. Specifically, this is to solve the following problem using *Bayesian optimization*,

$$\arg\max_{\mathbf{h}\in\mathcal{H}} \textsc{mrs}(\mathcal{D}, \mathbf{h}, p_{\mathrm{m}}, Q), \tag{4}$$

given a training data set $\mathcal{D}$, a cutoff threshold p_{m} and the number of random weights used in MRS. Importantly, as the architecture could shrink and expand in the search, its natural representation takes a variable-length form, which does not reconcile well with the state-of-the-art BO algorithm. To resolve this issue, three fixed-length encoding schemes are proposed to represent network architectures with variable sizes. Note that in this paper the search space $\mathcal{H}$ is determined by each encoding scheme (please see below). Also, we only employ the random forest model in the Bayesian optimization procedure (described in Algorithm 1) for the following reason: the design space of neural architecture comprises of integer/Boolean variables, which can be dealt with naturally by random forests. Gaussian process regression, which works over Euclidean spaces, is by default not applicable in this scenario. Although there are many recently endeavours in extending GPR's ability to handle the discrete and integer variables (e.g., [31]), it is not our major aim herein to compare the performance of Bayesian optimization when coupled with different surrogate models and hence we decided to choose the simplest random forest model to validate the proposed algorithm.

3.1. Encoding schemes

Assuming that the number of neurons per each layer is restricted to the range $[\underline{N}..\bar{N}]$, the number of layers is $m \in [\underline{M}..\bar{M}]$, and T denotes the maximum number of steps taken in back-propagation throughout time, three encoding schemes are proposed in this paper:

- **Plain**: the total length of this encoding is $m + 1$.

  $$\mathbf{h} = [h_1, h_2, \dots, h_m, l] \in \left(\{0\} \cup [\underline{N}..\bar{N}]\right)^m \times [1..T],$$

  where h_i is the number of neurons per each layer and l is the number of time steps. Note that h_i can take value zero, meaning there is no neuron in this layer and hence it is effectively dropped in the decoding procedure.
- **Flag**: the total length of this encoding is $2m + 1$.

  $$\mathbf{h} = [h_1, b_1, h_2, b_2, \dots, h_m, b_m, l] \in [\underline{N}..\bar{N}]^m \times \{0, 1\}^m \times [1..T],$$

where $b_i \in \{0, 1\}$ is the so-called "flag" that disables layer h_i if $b_i = 0$ when decoding such a representation to compute the actual architecture.
- **Size**: the total length of this encoding is $m + 2$.

  $$\mathbf{h} = [h_1, h_2, \dots, h_m, s, l] \in [\underline{N}..\bar{N}]^m \times [1..m] \times [1..T],$$

  where $s \leq m$ is the number of layers from the start of the representation that are considered in decoding, namely only $h_1, h_2, \dots, h_s$ are used to generate the actual architecture. (See Fig. 1.)

We shall use the notation $\texttt{code} \in \{\text{plain, flag, size}\}$ for the encoding scheme henceforth. In this manner, a fixed-length representation can be used to optimize variable-size architectures. For each case, in the decoding procedure, an output layer is appended to the RNN structure encoded in the search algorithm, to match the expected output dimension. Note that the activation function of the output layer has to be set according to the type of the task in each problem.

3.2. Decoding

It is worthwhile to note that the decoding procedure of all three representations is a *many-to-one mapping*. For instance, given a plain representation with a maximum of five layers ($m = 5$), $[h_1, h_2, 0, 0, h_5, l]$ and $[h_1, h_2, h_5, 0, 0, l]$ are representing exactly the same architecture. If $[h_1, h_2, h_5, 0, 0, l]$ has already been evaluated in the optimization process, then assessing the performance of $[h_1, h_2, 0, 0, h_5, l]$ is purely redundant. To determine the equivalence among representations, it is necessary to apply appropriate decoding functions for each type of representation:

$$\texttt{decode}(\mathbf{h}) = \begin{cases} \text{keep } h_i \text{ if } h_i > 0, i = 1, \dots, m. & \text{if the plain encoding} \\ \text{keep } h_i \text{ if } b_i = 1, i = 1, \dots, m. & \text{if the flag encoding} \\ \mathbf{h} \mapsto [h_1, h_2, \dots, h_s, l] & \text{if the size encoding} \end{cases} \quad (5)$$

As the decoding function is a many-to-one mapping, the BO algorithm could potentially propose the same architecture constantly (even with different representations before decoding), and hence the search efficiency would be drastically affected due to the following facts (1) the convergence of BO would be hampered as such an iteration (where the seen architecture is proposed again) makes no progress and the there is no information gain for the surrogate model therein, and (2) the same network architecture has to be evaluated again by MRS, which is wasteful even if MRS is much more efficient as compared the full network training. To cope with the former issue, it is important to avoid proposing the same architecture again as much as possible. In this study, we propose two alternative strategies which both rely on the definition of *"infeasibility"* (please see below) for representations:

- to set the MRS value of infeasible representations to the worst possible value (zero), which will be learned by the surrogate model underlying BO. Hence, the infeasible ones would not likely to be proposed by the surrogate model, or
- to use the original MRS values (as in Eq. (1)) and add constraints on the EI criterion to screen out infeasible representations. Note that in this case the surrogate model will be built on the original MRS values.

For the latter, the simplest solution is to maintain a lookup table to register the architectures (together with objective values) that are evaluated before.

Infeasible representation. Taking the plain encoding scheme as an example, a representation taking the form $[h_1, \dots, h_q, 0, \dots, 0, l]$ (where $h_i > 0$) shall be called *feasible*, e.g., $[h_1, h_2, h_5, 0, 0, l]$ is an infeasible representation when $m = 5$. $[h_1, h_2, h_5, 0, 0, l]$ represents the same architecture with the other 16 representations (by inserting two zeros at four different positions, e.g., $[h_1, h_2, 0, 0, h_5, l]$ and $[h_1, 0, h_2, 0, h_5, l]$). The other representations shall be called *"infeasible"*, which will be assigned with a fixed objective value that is worse than all the feasible solutions. Particularly, since we are maximizing MRS (which is a probability value), we set the penalized objective function value to be equal to zero. The rationale behind this treatment is that whenever the Bayesian optimization (BO) algorithm proposes an infeasible representation, the penalized objective function value will be learned by the surrogate model of BO and hence the chance of generating such representations will diminish gradually. In this manner, we are guiding the optimization process through the feasible ones and thus the search space is virtually reduced. Note that the BO algorithm still needs to make lots of infeasible trials before it stops proposing the infeasible ones, due to the large combinatorial space. It is conceptually better to directly avoid generating such representations by a constraint handling method (see below). The idea of defining the infeasible representation can be easily extended to the flag encoding scheme by *masking* h_i with b_i, i.e., replacing the value of h_i with a zero if b_i is equal to zero. However, this idea cannot be applied to the *size* encoding scheme.

Constraint handling. To avoid generating infeasible representations, we propose to assign penalty values to infeasible ones and to use a constraint handling method when proposing new candidate representations in BO. In addition, representations that are already evaluated will be also be penalized by the length of itself (the maximum penalty at line 4). For an infeasible representation that has *not* been evaluated (line 5), the number of zeros located before the last nonzero element is used as the penalty value. In line 7, the decoded representation is registered in a set L to check whether a representation has been evaluated before. The penalty value will be added to the EI criterion when proposing the candidate representations (see line 13 of Algorithm 1). As for the constraint handling, a *dynamic penalty* method is adopted here, where the penalty value will be scaled up with increasing iterations of BO. We choose the dynamic penalty here because it yields a relatively small penalty in the early phase of the search, allowing for exploring the infeasible regions within the search space, which is particularly critical to move between disconnected feasible regions. Also, as the search iteration increases, the penalty value will be enlarged to ensure a feasible solution as the outcome. In this manner, the following penalized infill criterion is used to propose candidate representations (instead of Eq. (3)):

$$\mathbf{h}^* = \operatorname*{arg\,max}_{\mathbf{h} \in \mathcal{H}} \mathrm{EI}(\mathbf{h}; \mathcal{M}) - Ct \cdot \textsc{penalty}(\mathbf{h}, X), \quad (6)$$

where (1) X is a set containing all evaluated solutions (not decoded), (2) t is the iteration counter of BO, and (3) $C = 0.5$ is a scaling factor. The intuition of this treatment is that the penalty value would have a large impact on the maximization of EI in the late stage, such that the probability of generating infeasible solutions becomes marginal. Also, the penalty value of $\mathbf{h}$ equals its length when it has been evaluated before, i.e., $\mathbf{h} \in X$, for avoiding proposing duplicated solutions, and otherwise, it is set to penalize $\mathbf{h}$ by the number of zeros preceding non-zero elements thereof, namely,

$$\textsc{penalty}(\mathbf{h}, X) = \begin{cases} \texttt{length}(\mathbf{h}), & \text{if } \mathbf{h} \in X \\ |\{h_i \colon \forall i \in [1..n-1] \\ \quad \big(h_i = 0 \cap \exists j \in [i+1..n](h_j = 1)\big)\big\}|, & \text{otherwise.} \end{cases} \quad (7)$$

3.3. A warm-start strategy

Within the Bayesian optimization algorithm, a surrogate model (e.g., a random forest) is used to learn the mapping from the evaluated solutions to the corresponding objective values. Typically, the Bayesian optimization starts with initializing the surrogate model by some randomly generated solutions. The basic idea of the so-called "warm-start" strategy is to augment the initial solutions by a set of infeasible solutions that can be generated before the optimization, such that the optimization process is started with a priori information. The infeasible solutions can be generated by randomly picking some components of a solution and setting them to zero for both the plain and flag encoding. Additionally, the objective value of those infeasible solutions is assigned with some default bad value (it is set to zero here since the MRS measure, which is the objective function of the architecture search, is bounded by zero from below), without the need to execute the MRS procedure. We anticipate that this warm-start strategy will add a bias in proposing the new candidate solutions in BO, steering the optimization process away from the infeasible solutions.

In all, the pseudo-code of the proposed approach is described in Algorithm 1. After creating the initial data set of BO (X, Y) using Latin Hypercube Sampling [32], the user can choose to turn on the generation of the warm-data prior to the optimization loop (lines 6-9). A set X' consisting of decoded representations is meant to track all the unique architectures that have been evaluated in MRS (line 11). In line 16, the constrained EI maximization is applied if the constraint method is enabled. The newly proposed solution $\mathbf{h}^*$ is decoded (line 20), after which we check if its decoded counterpart $\mathbf{h}^{*\prime}$ has been evaluated (line 21). If $\mathbf{h}^{*\prime}$ is not evaluated before (line 22-28), the feasibility of $\mathbf{h}$ is then checked and its objective value is set to zero in case of being infeasible (Otherwise, we evaluate its decoded representation $\mathbf{h}^{*\prime}$ in MRS (line 26)) If $\mathbf{h}^{*\prime}$ has been evaluated before, its objective value is looked up in the data set (X, Y) (line 30 and 31). The newly proposed candidate representation and its objective value are appended to BO's data set (X, Y) (lines 33 and 34). Afterwards, the random forest model is re-trained on the augmented data set (line 35).

Algorithm 1 Efficient Architecture Optimization for RNNs

```
1:  input: A data set $\mathcal{D}$, an encoding scheme code $\in$ {plain, flag, size}, the random forests algorithm RF, and the maximal iteration number $t_{\max}$.
2:  output: a full training RNN model
3:  $C \leftarrow 0.5, t \leftarrow 0, p_m \leftarrow 0.01, Q \leftarrow 100$
4:  Determine the search space $\mathcal{H}$ according to code
5:  Generate $X \subseteq \mathcal{H}$ using Latin Hypercube Sampling
6:  $Y \leftarrow \{\text{MRS}(\mathcal{D}, \text{decode}(\mathbf{h}), t, p_m, Q) : \mathbf{h} \in X\}$    ▷ evaluate $X$
7:  if "warm-start" is enabled then
8:      generate the warm data $(X_{\text{warm}}, Y_{\text{warm}})$    ▷ See Section 3.3
9:      $X \leftarrow X \cup X_{\text{warm}}, Y \leftarrow Y \cup Y_{\text{warm}}$
10: end if
11: $X' \leftarrow \{\text{decode}(\mathbf{h}) : \mathbf{h} \in X\}$    ▷ set of evaluated architectures
12: $\mathcal{M} \leftarrow \text{RF}(X, Y)$    ▷ surrogate model training
13: while $t < t_{\max}$ do
14:     if "constraint-handling" is enabled then
15:         $\mathbf{h}^* \leftarrow \arg\max_{\mathbf{h} \in \mathcal{H}} \text{EI}(\mathbf{h}; \mathcal{M}) - Ct \cdot \text{PENALTY}(\mathbf{h}, X)$    ▷ penalized EI
16:     else
17:         $\mathbf{h}^* \leftarrow \arg\max_{\mathbf{h} \in \mathcal{H}} \text{EI}(\mathbf{h}; \mathcal{M})$    ▷ unconstrained case
18:     end if
19:     $\mathbf{h}^{*\prime} \leftarrow \text{decode}(\mathbf{h}^*)$    ▷ solution decoding (Eq. (5))
20:     if $\mathbf{h}^{*\prime} \notin X'$ then    ▷ for unseen architectures
21:         if "infeasible-solution" is enabled and
22:             code $\neq$ size and $\mathbf{h}^*$ is infeasible then
23:             $y^* \leftarrow -\text{INF}$    ▷ penalty value for the infeasible ones
24:         else
25:             $y^* \leftarrow \text{MRS}(\mathcal{D}, \mathbf{h}^{*\prime}, t, p_m, Q)$    ▷ evaluate $\mathbf{h}^{*\prime}$ using MRS
26:         end if
27:         $X' \leftarrow X' \cup \{\mathbf{h}^{*\prime}\}$    ▷ add to the set of evaluated architectures
28:     else
29:         $S \leftarrow \{y : \forall(\mathbf{h}, y) \in (X, Y) \wedge \text{decode}(\mathbf{h}) = \mathbf{h}^{*\prime}\}$    ▷ the objective value of evaluated solutions that decodes to the same architecture as $\mathbf{h}^{*\prime}$
30:         $y^* \leftarrow$ sample a value from $S$ uniform at random
31:     end if
32:     $X \leftarrow X \cup \{\mathbf{h}^*\}, Y \leftarrow Y \cup \{y^*\}$    ▷ augment the data set
33:     $\mathcal{M} \leftarrow \text{RF}(X, Y)$    ▷ re-train the random forest model
34:     $t \leftarrow t + 1$
35: end while
36: $y_{\text{best}} \leftarrow \max\{Y\}$ and $\mathbf{h}_{\text{best}}$ is the corresponding solution to $y_{\text{best}}$
37: $\mathbf{h}_{\text{trained}} \leftarrow \text{ADAM}(\mathcal{D}, \mathbf{h}_{\text{best}})$    ▷ train the final neural architecture
38: return $\mathbf{h}_{\text{trained}}$
```

4. Experiments

In this section, we present the experimental study performed to test the proposed approach. First, we present the three prediction problems used to benchmark the method. Second, we present the experimental setup and the results of several combinations of the three strategies presented, i.e., infeasible solution, warm start, constraint handling, and encoding. Later, we compare the time between MRS and (short training) Adam. Finally, we study the error trade-off while changing the number of MRS samples.

4.1. Data sets

We tested the approach on three prediction problems: *sine wave*, *waste* [33], and *load forecast* [34].

The sine wave. Is a mathematical curve that represents a periodic oscillation. Despite its simplicity, it is extensively used to analyze systems [35]. It is usually expressed as a function of time (t), where A is the peak amplitude, f the frequency, and ϕ the phase (Eq. (8)). Its study is interesting because, by adding sine waves, it is possible to approximate any periodic waveform [35]. We sampled the sine wave described by: $A = 1, f = 1$, and $\phi = 0$, in the range $t \in [0, 100]$ seconds, and at 10 samples per second. Then, given a truncated part of the time series (i.e., a time steps number of points of the sampled sine wave), the problem consists in predicting the next value.

$$y(t) = A\sin(2\pi ft + \phi) \tag{8}$$

The waste *problem.* Introduced in [33], consists of predicting the filling level of 217 recycling bins located in the metropolitan area of a city in Spain, recorded daily for one year. Thus, given the historical filling levels of all containers (217 input values per day), the task is to predict the next day (i.e., the filling level of all bins). It is important to notice that this problem has been used as a benchmark in several studies [24,36,37] and that it is a real-world problem.

The load forecast *problem.* Provided by the European Network on Intelligent Technologies for Smart Adaptive Systems (EUNITE, http://www.eunite.org) as part of a competition [34,38], is a

Table 1
Optimization search spaces.

Parameter	Load range	Sine range	Waste range
Hidden layers (M)	[1, 8]	[1, 3]	[1, 8]
Look back (T)	[2, 30]	[2, 30]	[2, 30]
Neurons per layer (N)	[10, 100]	[1, 100]	[1, 300]

Table 2
BO and MRS parameter values.

Parameter	Value	Parameter	Value
No. samples (Q)	100	Threshold (p_m)	0.01
Max evaluations	100	Init solutions	10
Epochs	1000	Dropout	0.5

data set consisting of the electricity load demand of the Eastern Slovakian Electricity Corporation. It was recorded every half hour, from January 1, 1997, to January 31, 1999. Also, the temperature (daily mean) and the working calendar for this period are provided. Then, based on this data, the challenge is to predict the next maximum daily load. In other words, given the load demand (52 variables), i.e., the load demand recorded every half hour (48), the max daily load (1), the daily average temperature (1), the weekday (1), and the working day information (1), the task is to predict the max daily load of the next day (1). Note that the last month is used as the test data, thus our results may be compared directly against the competitors.

4.2. Performance

We implemented our approach[1] in Python 3, using DLOPT [39], MIP-EGO [40], Keras [41], and Tensorflow [42]. We used LSTM cells to build the decoded stacked architectures (as a way to mitigate the exploding and vanishing gradient problems [14]), and Adam truncated through time [43] (i.e., sharing all parameters in the unfolded models) to train the final solutions, with default parameter values [10].

We defined the search space (i.e., the constraints to the RNN architectures) of the three problems studied (Table 1) according to the datasets and the state-of-the-art. Particularly, the sine wave search space is taken from [24] and the waste search space is copied from [37] to enable a direct comparison.

Also, to ease the visualization of the results, we defined the following naming scheme to denote different combinations of encoding, warm start, invalid, and the constraint handling method: [constraint][warm start][infeasible][encoding].

Specifically, we use a character to denote each variant: Constraint (C), Warm start (w), Infeasible (I), and Encoding (F: flag, S: size, and P: plain). A dash (-) means that the corresponding alternative was not used. For example, -W-F corresponds to the combination of warm data and the flag encoding (i.e., without constraint handling and without invalid solution penalty).

Finally, we execute 30 independent runs for each combination of encoding, warm start, and the constraint handling method on a heterogeneous Linux cluster with more than 200 cores and 700 GB RAM, managed by HTCondor. In these experiments we used the optimization parameter values presented in Table 2. The remainder of this subsection introduces the performance results for the three problems and some insights into the solutions.

Note that the parameters presented in Tables 1 and 2 were taken from [24,37]. We decided to chose these values (instead of performing an hyperparameter tuning) to enable a direct comparison with our competitors.

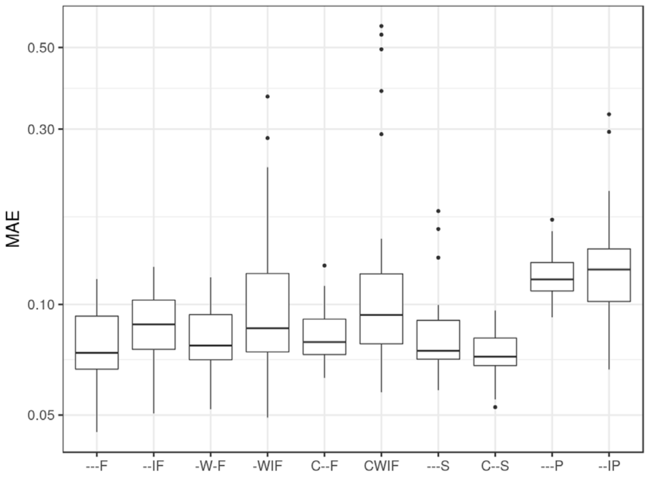


Fig. 2. MAE of the sine wave solutions.

4.2.1. Sine wave

The range of the sine function is [0, 1], thus we set the activation function of the dense output layer to be a tanh. Due to the immense number of *invalid* solutions, we implemented a *limited* version of the infeasible solution listing, i.e., instead of enumerating all infeasible solutions, we list a subset of them. Particularly, we listed the infeasible solutions described by the min and max values of each parameter (i.e., the number of neurons per layer and look back). Thus, we added 80 infeasible solutions to the warm-start.

Table 3 summarizes the results of the experiments, where MLES and GDET are the results presented in [24], and the other results correspond to the tested combinations. Fig. 2 shows the distribution of the MAE of the solutions of the sine wave problem. The Friedman rank sum test p-value is less than 2.2e−16 (chi-squared = 138.17, df = 11). Therefore, we performed a pairwise comparison using the Conover test for a two-way balanced complete block design [44], and the Holm p-value adjustment method. The results are presented in the row label Conover in Table 3. Groups sharing a letter are not significantly different ($\alpha = 0.01$).

The results show that using BO and MRS improves the performance of the final solution (error). On the other hand, multiple combinations of the proposed strategies (i.e., the combinations grouped by the letter d) show a similar performance.

4.2.2. Waste

The filling level of the bins ranges from 0 to 1. Accordingly, we set the activation function of the output layer to be a sigmoid. In this case, we added 126976 invalid solutions to the warm start.

Table 4 summarizes the results of the tests on the waste problem. The table also includes the results of [37] (Cities) and [24] (MLES). Fig. 3 shows the distribution of the MAE of the solutions of the waste problem. The Friedman rank sum test p-value is equal to 0.02401 (chi-squared = 22.048, df = 11). Therefore, we performed a pairwise comparison using the Conover test for a two-way balanced complete block design , and the Holm p-value adjustment method. The results are presented in the row label Conover in Table 4. Groups sharing a letter are not significantly different ($\alpha = 0.01$).

In this case, our results are as good as our competitors (the results grouped by the letter a). Nonetheless, it is important to remark that [37] (Cities) trains every candidate solution using Adam, turning out to be more time-consuming.

[1] Code available in https://github.com/acamero/dlopt.

Table 3
Sine optimization results (MAE of the best solution). Groups sharing a letter in the Conover row are not significantly different.

	GDET	MLES	—F	–IF	-W-F	-WIF	C–F	CWIF	—S	C—S	—P	–IP
Mean	0.1419	0.1047	0.0785	0.0882	0.0816	0.1119	0.0839	0.1452	0.0857	0.0745	0.1198	0.1363
Median	0.1489	0.0996	0.0738	0.0882	0.0772	0.0861	0.0789	0.0935	0.0748	0.0721	0.1170	0.1244
Max	0.2695	0.2466	0.1172	0.1266	0.1185	0.3677	0.1276	0.5723	0.1794	0.0962	0.1700	0.3290
Min	0.0540	0.0631	0.0449	0.0505	0.0518	0.0492	0.0631	0.0577	0.0584	0.0525	0.0922	0.0665
Sd	0.0513	0.0350	0.0194	0.0182	0.0161	0.0695	0.0154	0.1367	0.0274	0.0109	0.0177	0.0558
Conover	a	bc	**d**	ef	**d**e	bf	**d**ef	c	**d**ef	**d**	a	a

Table 4
Waste optimization results (MAE of the final solution). Groups sharing a letter in the Conover row are not significantly different.

	Cities	MLES	—F	–IF	-W-F	-WIF	C–F	CWIF	—S	C—S	—P	–IP
Mean	0.0728	0.0790	0.0722	0.0821	0.0730	0.0812	0.0728	0.0728	0.0732	0.0725	0.0744	0.0735
Median	0.0731	0.0728	0.0723	0.0735	0.0725	0.0730	0.0725	0.0725	0.0736	0.0723	0.0737	0.0731
Max	0.0757	0.1377	0.0791	0.1227	0.0806	0.1231	0.0767	0.0767	0.0756	0.0760	0.0920	0.0883
Min	0.0709	0.0691	0.0695	0.0691	0.0703	0.0698	0.0692	0.0701	0.0691	0.0688	0.0717	0.0701
Sd	0.0012	0.0172	0.0019	0.0156	0.0020	0.0177	0.0018	0.0014	0.0015	0.0016	0.0041	0.0027
Conover	**a**bc	**a**bc	**a**	bcd	**a**b	d	**a**b	**a**b	bcd	**a**b	cd	bcd

Table 5
Optimization results (MAPE of the final solution). Groups sharing a letter in the Conover row are not significantly different.

	SVM	RBF	WK+	—F	–IF	-W-F	-WIF	C–F	CWIF	—S	C—S	—P	–IP
Mean	2.879	NA	NA	2.726	3.148	2.595	3.066	2.158	2.844	2.321	2.235	4.823	5.287
Median	2.945	NA	NA	2.466	2.933	2.368	2.814	2.099	2.846	2.125	2.050	5.040	5.213
Max	3.480	NA	NA	6.271	5.207	4.594	6.031	3.364	3.901	6.271	4.605	6.999	11.004
Min	1.950	1.481	1.323	1.840	1.759	1.593	1.919	1.452	2.033	1.654	1.657	3.142	3.415
Sd	0.004	NA	NA	0.888	1.013	0.765	1.000	0.440	0.515	0.774	0.564	0.884	1.727
Conover	NA	NA	NA	abc	a	bc	a	**d**	ab	ce	**d**e	f	f

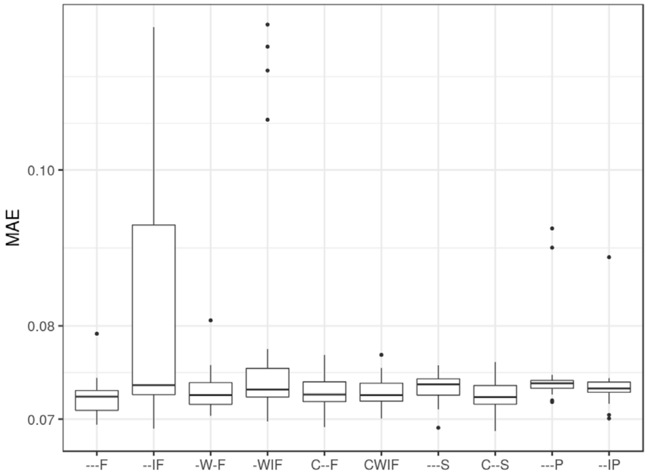


Fig. 3. MAE waste.

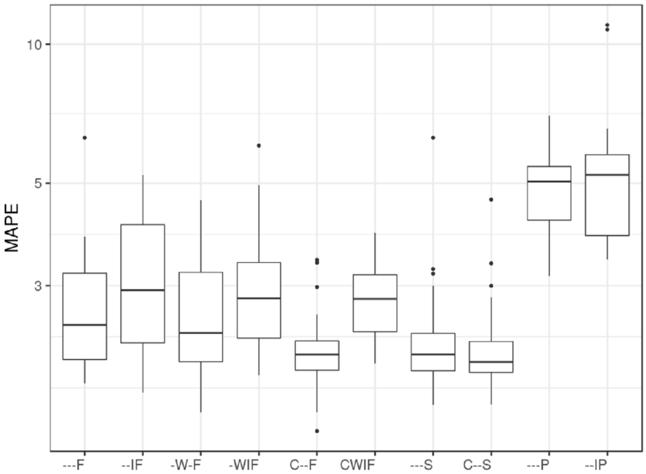


Fig. 4. MAPE load forecast.

4.2.3. *Load forecast*

According to the preprocessing performed in [38], we normalized the data to have a mean equal to zero and a standard deviation equal to one. Then, we set the activation function of the output layer to be `linear`. Besides, we added 126976 invalid solutions to the warm start.

Table 4 summarizes our results and the ones presented in [34] (SVM), and [38] (RBF and WK+, WKNNRW in the original work). In this case, we present the mean absolute percentage error (MAPE) because it is the performance metric used in the referred studies (`NA` indicates the corresponding data is not available). Fig. 4 shows the distribution of the MAPE of the solutions of the waste problem. Unfortunately, in this case, we do not have the detailed results of SVM, RBF, and WK+. Thus, we cannot perform a detailed analysis considering all competitors. Nonetheless, we performed a detailed analysis considering exclusively the results of our tests. The Friedman rank sum test p-value is less than 2.2×10^{-16} (chi-squared = 146.38, df = 9). Therefore, we performed a pairwise comparison using the Conover test for a two-way balanced complete block design, and the Holm p-value adjustment method. The results are presented in the row label Conover in Table 5. Groups sharing a letter are not significantly different ($\alpha = 0.01$).

4.2.4. *Solutions overview*

To get insights into the RNN architectures, we analyzed the (best) solutions. Fig. 5 shows the percentage of solutions that have a specific number of hidden layers (within the search space defined in Table 1). Fig. 6 presents the percentage of solutions that have each of the possible look backs. Fig. 7 depicts the distribution of the total number of LSTM cells.

It is no surprise that the *plain* encoding produced deeper and bigger (in terms of the total number of neurons) solutions,

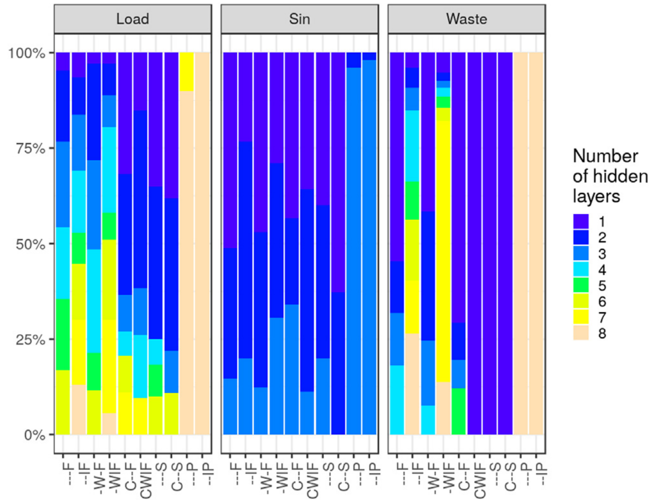


Fig. 5. Number of hidden layers of the solutions.

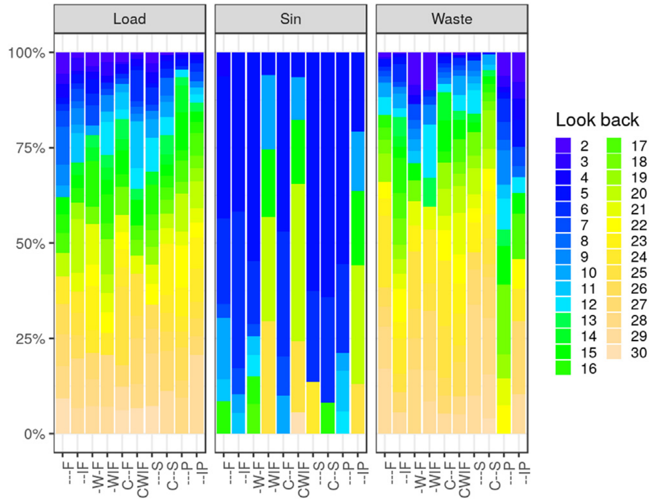


Fig. 6. Look back or time steps of the solutions.

because of its own encoding limitations. On the other hand, two relatively similar combinations in terms of the error, namely C--F and C--S, present different architecture combinations.

Also, it is quite interesting that there is no clear *architecture trend*. There are some value ranges that seem to be more suitable, e.g., shallower instead of deeper networks, or mid-to-upper look back values for the load forecast problem, but we cannot conclude that there is an *all-rounder* architecture.

4.3. Time analysis

The results presented in this study (Table 4) show that using MRS as a proxy of the performance is as good as using short training results. However, as it is claimed in [6], MRS is supposed to be a low-cost approach. Therefore, we compared the run time of Adam against MRS. Specifically, we randomly select 16 runs from the previous experiments (i.e., 100 architectures evaluated in 16 runs, totaling 1600 RNNs). Then, for each network we performed a MRS (100 samples) and a 10 epochs training using Adam.

We repeated the experiments because of two reasons. First, the previous experiments were run on a cluster of heterogeneous computers (hence the run times were not fairly comparable). Secondly, the final solutions were trained for 1000 epochs, thus the comparison would not have been fair.

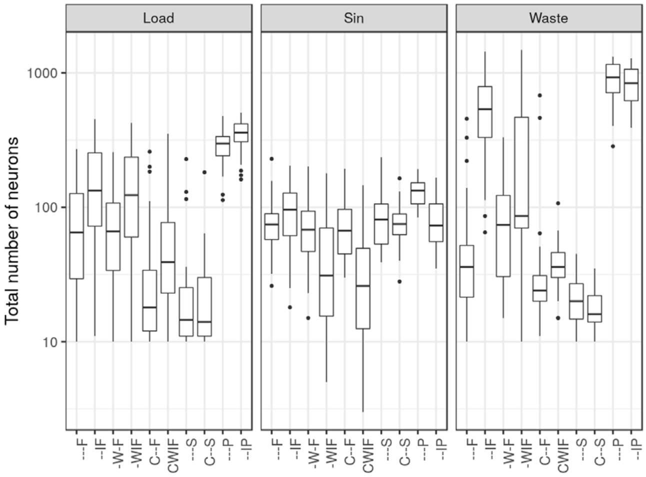


Fig. 7. Distribution of the total number of LSTM cells.

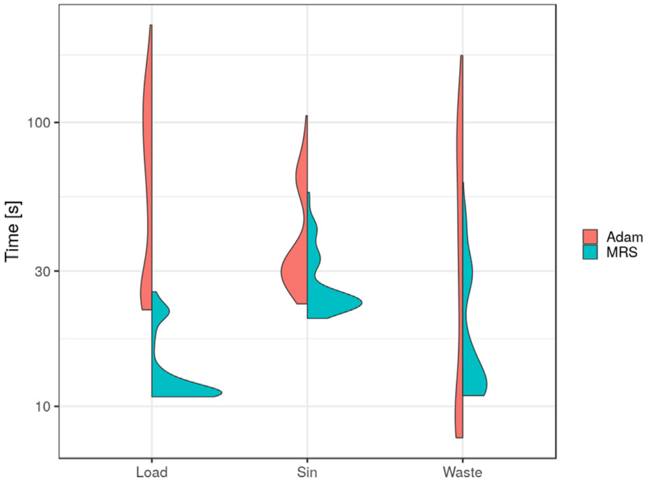


Fig. 8. Time comparison: Adam (10 epochs) vs MRS (100 samples).

Table 6 summarizes the time in seconds for both approaches, and Fig. 8 shows the distribution of the time (in seconds). We performed a Wilcoxon rank sum test to compare both approaches. Note that we compare the overall results and the results of each problem independently. The results are presented in the table (*Signif.*) using the following codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1.

On average, MRS is 2.6 times faster than Adam. These results are in line with the ones presented in [24]. In other words, if we have used the results of 10 epochs training using Adam to compare the architectures during the optimization process (instead of MRS), we will have spent more than twice the time!

4.4. Error trade-off

Moreover, we studied how much the outcome of MRS is affected (i.e., error of the final solution) when the number of samples is changed. We repeated the *waste* and *load forecast* experiments using the C--S configuration, and 30, 50, and 200 samples per each solution evaluated (MRS).

Table 7 summarizes the error trade-off results. The Friedman rank sum test *p*-value is equal to 0.004996 (chi-squared = 12.84,

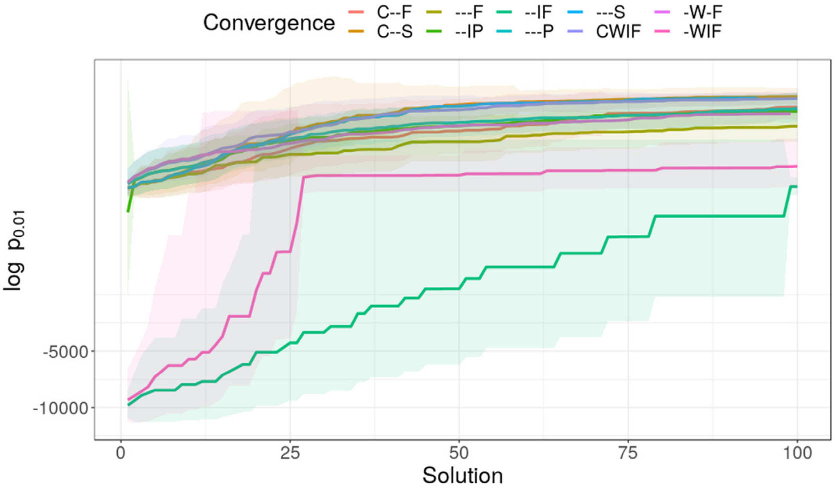


Fig. 9. Average convergence of the fitness of the solutions for the waste problem.

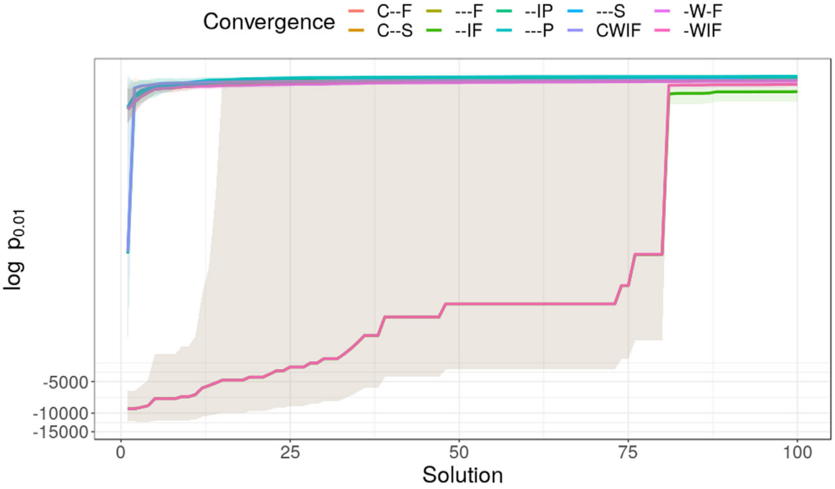


Fig. 10. Average convergence of the fitness of the solutions for the load forecast problem.

Table 6
Time comparison in seconds: Adam vs MRS. According to the Wilcoxon rank sum test, there is a significant improvement.

	[seconds]	Load	Sin	Waste	Overall
Adam	Mean	72.1	41.8	45.3	53.1
	Median	62.6	32.3	29.1	34.1
	Max	220.9	105.8	172.3	220.9
	Min	21.9	23.0	7.8	7.8
	Sd	48.7	19.0	40.8	40.5
MRS	Mean	13.8	27.9	19.3	20.3
	Median	11.7	23.9	13.8	20.0
	Max	25.3	56.8	61.6	61.6
	Min	10.8	20.4	10.9	10.8
	Sd	4.3	8.0	10.7	10.0
Signif. (Adam vs MRS)		***	***	***	***

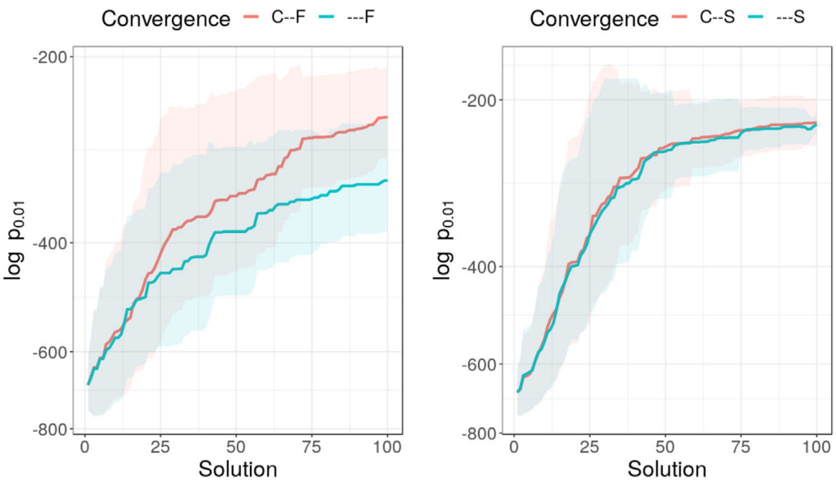
df = 3) in the waste problem, while it is equal to 0.0003184 (chi-squared = 18.68, df = 3) in the load forecast problem. Therefore, we performed a pairwise comparison using the Conover test for a two-way balanced complete block design [44], and the Holm p-value adjustment method. The results are presented in the row Conover of both tables. Groups sharing a letter are not significantly different ($\alpha = 0.01$).

The results show that we might reduce the time (by taking fewer samples) but with an error increase. On the other hand, doubling the number of samples (used in this study), we will have not reduced the error. Nonetheless, it is quite interesting that even with a small number of samples, lets say 30, it is possible to estimate the performance of a network.

4.5. Algorithm convergence

Finally, we studied the convergence of the proposed algorithm. Particularly, we analyzed the fitness (probability estimated by the MRS) of the solutions as the search was done. Figs. 9 and 10 depict the best-so-far MRS value against the number of candidates evaluated, average over all independent runs for each combination of encoding, warm-start, and constraint handling methods (shown by the bold line). Also, the standard deviation is illustrated by the shaded areas. It is important to point out that a higher MRS value is correlated with a better performance after training the network using Adam [6], hence indicating that all combinations are converging.

Moreover, to show the impact of the penalty function, we compared the pairs `C--S` and `---S`, `C--F` and `---F`. Notice that the results present the average value of the MRS and the standard deviation (shaded area) for 30 independent runs (each combination) in the waste prediction problem. Therefore, we assume that the difference in performance (i.e., the convergence) can be explained by the penalty. (See Fig. 11.)



(a) (b)

Fig. 11. Impact of the penalty on the average convergence (waste problem).

Table 7
Waste and Load trade-off results. Groups sharing a letter in the Conover row are not significantly different.

	Samples	30	50	100	200
Waste (MAE)	Mean	0.0734	0.0734	0.0725	0.0723
	Median	0.0732	0.0740	0.0723	0.0726
	Max	0.0778	0.0780	0.0760	0.0757
	Min	0.0694	0.0690	0.0688	0.0685
	Sd	0.0017	0.0020	0.0016	0.0018
Load (MAPE)	Mean	2.664	2.616	2.235	2.137
	Median	2.510	2.555	2.050	2.073
	Max	4.436	3.750	4.605	3.146
	Min	1.930	1.884	1.657	1.521
	Std	0.597	0.492	0.564	0.405
Conover		a	a	b	b

5. Conclusions and future work

In this study, we propose to optimize the architecture of a recurrent neural network with a combination of Bayesian optimization and Mean Absolute Error Random Sampling (MRS). More specifically, we propose three fixed-length encoding schemes to represent variable size architectures (*flag*, *plain*, and *size*), an alternative to deal with the many-to-one problem derived from the fixed-variable-length problem (i.e., the *infeasible*solution), and two strategies to cope with the fixed-variable-length problem, namely *warm-start* and *constraints handling*.

We test our proposal on three prediction problems: the sine wave, the waste filling level of 217 bins in a metropolitan area of a city in Spain, and the maximum daily load forecast of an electricity company in Slovakia. We benchmark our proposal against state-of-the-art techniques, and we performed a time comparison and an error trade-off study. Notice that for each problem a different activation function has been used, namely, `tanh`, `sigmoid`, and `linear`.

The results show that none of the strategies presented outperforms the others in all cases. Nonetheless, using the *size* encoding and the *constraints handling* consistently show to be an *effective* alternative to the problem.

Moreover, the results show that MRS is an *efficient* alternative to optimize the architecture of an RNN. Particularly, we showed that evaluating an architecture using MRS is 2.6 times faster than performing a short training (ten epochs) using Adam, and without losing performance.

Overall, using BO, in combination with MRS, shows to be a competitive approach to optimize the architecture of an RNN. It offers a state-of-the-art error performance, while the time is drastically reduced.

Finally, for the next step, several issues have to be addressed. First, it is necessary to test on more data sets to validate the proposal. Second, MRS has to be further researched because it shows to be a promising alternative, but there is no clear explanation of why it works. Additionally, it will be interesting to use the *warm start* strategy to explore *augmenting restarts*, i.e., iteratively increase the number of hidden layers and feeding the model with the previous results.

CRediT authorship contribution statement

Andrés Camero: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing - original draft. **Hao Wang:** Conceptualization, Methodology, Software, Validation, Writing - original draft. **Enrique Alba:** Conceptualization, Writing - review & editing, Funding acquisition, Resources. **Thomas Bäck:** Conceptualization, Writing - review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by Universidad de Málaga, Andalucía Tech, Spain, Consejería de Economía y Conocimiento de la Junta de Andaluía, Ministerio de Economía, Industria y Competitividad, Gobierno de España, Spain, and European Regional Development Fund grant numbers TIN2017-88213-R (6city.lcc.uma.es), RTC-2017-6714-5 (ecoiot.lcc.uma.es), and UMA18-FEDERJA-003 (Precog). And by the Helmholtz Association's Initiative and Networking Fund (INF), Germany under the Helmholtz AI platform grant agreement (ID ZT-I-PF-5-1).

References

[1] S. Haykin, Neural Networks and Learning Machines, volume 3, Pearson Upper Saddle River, NJ, USA:, 2009.

[2] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.

[3] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 157–166.

[4] V.K. Ojha, A. Abraham, V. Snášel, Metaheuristic design of feedforward neural networks: A review of two decades of research, Eng. Appl. Artif. Intell. 60 (2017) 97–116.

[5] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, JMLR.org, 2013, pp. III–1310–III–1318.

[6] A. Camero, J. Toutouh, E. Alba, Low-cost recurrent neural network expected performance evaluation, 2018, Preprint arXiv:1805.07159.

[7] T. Domhan, J.T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in: Proceedings of the 24th International Conference on Artificial Intelligence, AAAI Press, 2015, pp. 3460–3468.

[8] D.R. Jones, M. Schonlau, W.J. Welch, Efficient global optimization of expensive black-box functions, J. Global Optim. 13 (4) (1998) 455–492.

[9] J. Močkus, On Bayesian methods for seeking the extremum, in: Optimization Techniques IFIP Technical Conference, Springer, 1975, pp. 400–404.

[10] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[11] J.S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 24, Curran Associates, Inc., 2011, pp. 2546–2554.

[12] A. Camero, J. Toutouh, D.H. Stolfi, E. Alba, Evolutionary deep learning for car park occupancy prediction in smart cities, in: Intl. Conf. on Learning and Intelligent Optimization, Springer, 2018, pp. 386–401.

[13] R. Jozefowicz, W. Zaremba, I. Sutskever, An empirical exploration of recurrent network architectures, in: Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, JMLR.org, 2015, pp. 2342–2350.

[14] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[15] E.Z. Ramos, M. Nakakuni, E. Yfantis, Quantitative measures to evaluate neural network weight initialization strategies, in: 2017 IEEE 7th Annual Computing and Communication Workshop and Conference, CCWC, IEEE, 2017, pp. 1–7.

[16] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks, J. Mach. Learn. Res. 10 (Jan) (2009) 1–40.

[17] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.

[18] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al., Evolving deep neural networks, in: Artificial Intelligence in the Age of Neural Networks and Brain Computing, Elsevier, 2019, pp. 293–312.

[19] A. Rawal, R. Miikkulainen, Evolving deep lstm-based memory networks using an information maximization objective, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, 2016, pp. 501–508.

[20] J. Liang, E. Meyerson, R. Miikkulainen, Evolutionary architecture search for deep multitask networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2018, pp. 466–473, http://dx.doi.org/10.1145/3205455.3205489, URL: http://doi.acm.org/10.1145/3205455.3205489.

[21] A. ElSaid, S. Benson, S. Patwardhan, D. Stadem, T. Desell, Evolving recurrent neural networks for time series data prediction of coal plant parameters, in: Intl Conf on the Applications of Evolutionary Computation, Part of EvoStar, Springer, 2019, pp. 488–503.

[22] A. Ororbia, A. ElSaid, T. Desell, Investigating recurrent neural network memory structures using neuro-evolution, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2019, pp. 446–455.

[23] A. ElSaid, F.E. Jamiy, J. Higgins, B. Wild, T. Desell, Using ant colony optimization to optimize long short-term memory recurrent neural networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2018, pp. 13–20, http://dx.doi.org/10.1145/3205455.3205637, URL: http://doi.acm.org/10.1145/3205455.3205637.

[24] A. Camero, J. Toutouh, E. Alba, Random error sampling-based recurrent neural network architecture optimization, Eng. Appl. Artif. Intell. 96 (2020) 103946.

[25] T. Bartz-Beielstein, C.W.G. Lasarczyk, M. Preuss, Sequential parameter optimization, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 1, 2005, pp. 773–780, http://dx.doi.org/10.1109/CEC.2005.1554761.

[26] D. Horn, T. Wagner, D. Biermann, C. Weihs, B. Bischl, Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark, in: Evolutionary Multi-Criterion Optimization, Springer, 2015, pp. 64–78.

[27] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 507–523.

[28] H. Wang, B. van Stein, M. Emmerich, T. Bäck, A new acquisition function for Bayesian optimization based on the moment-generating function, in: 2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC, 2017, pp. 507–512, http://dx.doi.org/10.1109/SMC.2017.8122656.

[29] J. Kim, M. McCourt, T. You, S. Kim, S. Choi, Bayesian optimization over sets, in: 6th ICML Workshop on Automated Machine Learning, 2019.

[30] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N. De Freitas, Taking the human out of the loop: A review of Bayesian optimization, Proc. IEEE 104 (1) (2015) 148–175.

[31] D. Nguyen, S. Gupta, S. Rana, A. Shilton, S. Venkatesh, Bayesian Optimization for categorical and category-specific continuous inputs, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, the Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, the Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 5256–5263, URL: https://aaai.org/ojs/index.php/AAAI/article/view/5971.

[32] M.D. McKay, R.J. Beckman, W.J. Conover, Comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (2) (1979) 239–245.

[33] J. Ferrer, E. Alba, BIN-CT: Sistema inteligente para la gestión de la recogida de residuos urbanos, in: International Greencities Congress, 2018, pp. 117–128.

[34] B.-J. Chen, M.-W. Chang, et al., Load forecasting using support vector machines: A study on EUNITE competition 2001, IEEE Trans. Power Syst. 19 (4) (2004) 1821–1830.

[35] R.N. Bracewell, R.N. Bracewell, The Fourier Transform and Its Applications, Vol. 31999, McGraw-Hill New York, 1986.

[36] J. Ferrer, E. Alba, BIN-CT: Urban waste collection based on predicting the container fill level, Biosystems (2019).

[37] A. Camero, J. Toutouh, J. Ferrer, E. Alba, Waste generation prediction under uncertainty in smart cities through deep neuroevolution, Rev. Fac. Ing. (2019).

[38] K. Lang, M. Zhang, Y. Yuan, X. Yue, Short-term load forecasting based on multivariate time series prediction and weighted neural network with random weights and kernels, Cluster Comput. (2018) 1–9.

[39] A. Camero, J. Toutouh, E. Alba, DLOPT: Deep learning optimization library, 2018, arXiv preprint arXiv:1807.03523.

[40] H. Wang, M. Emmerich, T. Bäck, Cooling strategies for the moment-generating function in Bayesian global optimization, in: 2018 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2018, pp. 1–8.

[41] F. Chollet, et al., Keras, 2015, https://keras.io.

[42] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., TensorFlow: A system for large-scale machine learning, in: OSDI, 2016, pp. 265–283.

[43] P.J. Werbos, Backpropagation through time: What it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.

[44] W.J. Conover, R.L. Iman, On multiple-comparisons procedures, Los Alamos Sci. Lab. Tech. Rep. LA-7677-MS 1, 1979, p. 14.