Random Error Sampling-based Recurrent Neural Network Architecture Optimization

Andrés Camero[a,b,*], Jamal Toutouh[a,c], Enrique Alba[a]

[a] *Universidad de Málaga, ITIS Software, España*
[b] *GermanAerospace Center (DLR), Remote Sensing Technology Institute (IMF), Germany*
[c] *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, MA, USA*

Abstract

Recurrent neural networks are good at solving prediction problems. However, finding a network that suits a problem is quite hard because their performance is strongly affected by their architecture configuration. Automatic architecture optimization methods help to find the most suitable design, but they are not extensively adopted because of their high computational cost. In this work, we introduce the Random Error Sampling-based Neuroevolution (RESN), an evolutionary algorithm that uses the mean absolute error random sampling, a training-free approach to predict the expected performance of an artificial neural network, to optimize the architecture of a network. We empirically validate our proposal on four prediction problems, and compare our technique to training-based architecture optimization techniques, neuroevolutionary approaches, and expert designed solutions. Our findings show that we can achieve state-of-the-art error performance and that we reduce by half the time needed to perform the optimization.

Keywords: neuroevolution, metaheuristics, recurrent neural network, evolutionary algorithm

1. Introduction

Deep learning (DL) and deep neural networks (DNN) [1] are *everywhere*, improving state-of-the-art results of problems from a wide range of topics, from natural language to image processing and pattern recognition [2, 3].

There are several types of DNNs, where each one is suited for solving a specific problem. Among these network types, Recurrent Neural Networks (RNNs) are especially good at solving sequential modeling and prediction, e.g., natural language, and speech recognition and modeling [1]. Basically, RNNs are feed-forward networks that include feedback connections between layers and neurons, and this recurrence allows them to capture long-term dependency in the input. In spite of their great performance, RNNs have a drawback: they are hard to train, because of the *vanishing* and the *exploding* gradient problems [4, 5].

An alternative to mitigate these problems is to optimize the architecture of the network. By selecting an appropriate configuration of the parameters of the network (e.g., the activation functions, the number of hidden layers, the kernel size of a layer, etc.), it is tailored to the problem and by this means the performance is improved [6, 7, 8].

DNN architecture optimization methods can be grouped into two main groups: the manual exploration-based approaches (usually) lead by expert knowledge, and the automatic search-based methods, e.g., grid, neuroevolutionary or random search [9].

The number of alternatives or parameters to configure a DNN is extremely large. Thus the architecture optimization has to deal with a high-dimensional search space. Despite this size, most methods (manual and automatic) are based on *trial-and-error*, meaning that each architecture is trained (e.g., using a gradient-based algorithm) and tested to evaluate its numerical accuracy. Thus, the high-dimensional search space and the high cost of the evaluation limit the interest in this methodology [9].

Some authors have explored different approaches to speed up the evaluation of DNN architectures to improve the efficiency of automatic architecture optimization algorithms [10, 11]. Among them, the *Mean absolute error random sampling* (MRS) [10, 12] poses a different way of dealing with the problem. The main idea behind this method, inspired by the linear time-invariant theory, is to infer the numerical accuracy of a given network without actually training it. Given an input, several sets of random weights are generated and analyzed measuring the mean absolute error (MAE). Then, the probability of finding a set of weights whose MAE is below a predefined threshold is estimated.

Therefore, can we combine the MRS (to speed-up the search) and a gradient-based technique (to improve the performance) into a *hybrid* technique to optimize the architecture of an RNN? More specifically, we pose the following research questions:

RQ1 Can a *hybrid* (MRS and gradient-based) architecture opti-

*Correspondent author
Email addresses:
`andrescamero@uma.es`, `andres.camerounzueta@dlr.de` (Andrés Camero), `toutouh@mit.edu` (Jamal Toutouh), `ea@lcc.uma.es` (Enrique Alba)

mization technique get the same error performance of a solely gradient-based one?

RQ2 Can a *hybrid* architecture optimization technique get the same error performance of a non-gradient-based approach (.e.g., neuroevolutionary algorithm)?

RQ3 Can we reduce the architecture optimization time by using this *hybrid* approach?

RQ4 Can we improve the performance of an expert designed architecture/solution?

To answer this questions, we propose a novel technique: the Random Error Sampling-based Neuroevolution (RESN), an evolutionary algorithm (EA) that navigates through the architecture space and guides its search by using the MRS, avoiding the high-cost of training each candidate solution. Then, once the algorithm has computed a "final" solution, RESN will train it using a gradient descent-based method.

Therefore, the main contribution of this work is to propose a *hybrid* approach to optimize the architecture of a RNN, i.e., an evolutionary algorithm that relies on the MRS as a fitness measure to select an RNN architecture, and uses a gradient-based technique to train the *final* architecture.

The remainder of this paper is organized as follows: the next section briefly reviews related works. Section 3 introduces RESN. Section 4 presents the experimental design. Section 5 presents the results. And finally, Section 6 outlines the conclusions and proposes future work.

2. Related Work

This section outlines the related work. First, we introduce RNN architecture optimization and evaluation works. Second, we briefly review the MRS. Finally, we review the state-of-the-art of metaheuristics applied to DL optimization.

2.1. RNN Architecture Optimization

An RNN is an artificial neural network that adds recurrent (or feedback) edges that may form cycles and self connections [13]. Due to this recurrency, most gradient-based optimization procedures fail to train an RNN. The main issue with gradient-based approaches is that they keep a vector of activations, which makes RNNs extremely deep and aggravates the exploding and the vanishing gradient problems [4, 5, 14].

To tackle these gradient-related problems, Hochhreiter and Schmidhuber proposed an effective solution: the Long Short-Term Memory (LSTM) cell. The LSTM is a special *neuron design* that contains units called memory blocks in the recurrent hidden layer [15]. Despite LSTM effectively mitigates gradient problems (i.e., they are easier to train than standard RNNs), not only the network architecture affects the learning process but also the weight initialization [16] and all specific parameters of the optimization algorithm [17].

Therefore, to cope with the learning process as a whole, some authors have proposed to perform an architecture optimization [6, 7, 8]. Specifically, they propose to look for a specific architecture (the number of layers, the number of hidden unit per layer, etc.) and a set of parameters to train the network that improve the performance of the *optimized* network given a data set. In other words, instead of using a general configuration, the idea is to tailor the architecture to the problem.

When dealing with an architecture configuration, an expert can discard a configuration based on his expertise, i.e., without the need of evaluating it. However, intelligent automatic architecture optimization procedures search more efficiently through a high-dimensional search space.

Even though intelligent methods are more competitive than experts, they are not generally adopted because they are computationally intensive [9]. They require to fit a model and to evaluate its performance on validation data (i.e., they are data-driven), which can be a demanding process (time and computational resources) [6, 18, 19].

Hence, few methods have been proposed to address this issue by speeding up the evaluation of the proposed architecture. For example, Domhan et al. [11] proposed to predict the performance of a network based on the learning curve, reducing the search time up to 50%.

More recently, Camero et al. [10] presented the MRS, a novel low-cost method to compare the performance of RNN architectures without training them. Particularly, the MRS evaluates a candidate architecture by generating a set of random weights and evaluating its performance.

In line with the latter approach, we propose to use the MRS to guide an EA to search for the most suitable architecture.

2.2. Mean Absolute Error Random Sampling

At a glance, the MRS predicts how *easy* would be to train a network (i.e., the performance) by taking a user-defined number of samples of the output (on a given input) of a specific architecture. Each sample is taken using a (new) random normally distributed set of weights, and calculating the MAE. Then, a truncated normal distribution is fitted to the MAE values sampled, and a probability p_t of finding a set of weights whose error is below a user-defined threshold is estimated. Then, the probability p_t is used as a predictor of the performance (error) of the analyzed architecture.

Algorithm 1 (taken from [10]) presents the pseudo-code of the MRS. Given an architecture (*ARQ*), a number of time steps (or look back, *LB*), and a user-defined input (*data*), the algorithm initializes the architecture (**InitializeRNN**). Then, it takes *MAX_SAMPLES* samples of the MAE, i.e., for each sample a set of normally distributed weights is generated (function **GenerateNormalWeights**), with mean equal to zero, and standard deviation equal to one. The RNN is updated with the new set of weights (**UpdateWeights**), and the MAE is computed for the data using the updated RNN (**MAE**). Once the sampling is done, a truncated normal distribution is fitted to the MAE values sampled (**FitTruncatedNormal**), and finally, the probability p_t is estimated for a defined *THRESHOLD* (**PTruncatedNormal**).

The details and the design considerations, including the tuning of the parameters, are thoroughly discussed in the original proposal [10].

Algorithm 1 MRS pseudo-code [10]

Given: an architecture (*ARQ*), a number of time steps or look back (*LB*), a user-defined time series (*data*), a number of samples (*MAX_SAMPLES*), and a *THRESHOLD*.
rnn ← InitializeRNN(ARQ, LB)
mae ← ∅
while sample ≤ MAX_SAMPLES **do**
    weights ← GenerateNormalWeights($\mu = 0, \sigma = 1$)
    UpdateWeights(rnn, weights)
    mae[sample] ← MAE(rnn, data)
    sample + +
end while
mean, sd ← FitTruncatedNormal(mae)
p_t ← PTruncatedNormal(mean, sd, THRESHOLD)

2.3. Deep Learning and Metaheuristics

Metaheuristics are well-known optimization algorithms to address complex, non-linear, and non-differentiable problems [9, 20]. They efficiently combine exploration and exploitation strategies to provide *good* solutions requiring bounded computational resources.

Optimization in DL may be viewed from different perspectives: training as optimization of the DNN weights, hyperparameter selection, network topologies, learning environment, etc. These different points of view are adopted to improve the DNNs generalization capabilities.

Gradient-descent based methods, such as back-propagation, are widely used to train DNNs. However, these methods need several manual tuning schemes to make their parameters optimal and it is difficult to parallelize them taking advantage of graphics processing units (GPUs). Thus, several authors have explored DNNs training by using metaheuristics, an idea explored long before DNN rise [21, 22]. Different authors combined convolutional neural networks with metaheuristics to improve their accuracy and performance by optimizing the layers weights and threshold. Following this idea, Zhining and Yunming [23] used a genetic algorithm (GA); Rosa et al. [24] applied harmony search (HS); Rere et al. [25] analyzed simulated annealing (SA), and later the same authors evaluated SA, differential evolution (DE), and HS [26].

Some authors also explored coupling training based on stochastic gradient descent (SGD) with metaheuristics. This approach has provided promising results in training generative adversarial networks (GANs) GANs combine a generative network (generator) and a discriminative network (discriminator) that apply adversarial learning to be trained. Evolutionary GAN (E-GAN) applies ES to evolve a population of networks (generators) that are mutated by applying SGD according to different loss functions [27]. The generators are evaluated by a single discriminator that returns a fitness value for each generator. Lipizzaner [28] and Mustangs [29] are competitive coevolutionary algorithms that evolve two populations, one of generators and one of discriminators, to improve diversity during the training. They also apply SGD-based mutation to generate the offspring.

GA has been applied to evolve increasingly complex neural network topologies and the weights simultaneously, in the NeuroEvolution of Augmenting Topologies (NEAT) method [30, 31]. However, NEAT has some limitations when it comes to evolving DNNs and RNNs [32].

Focusing on RNNs, NEAT-LSTM [33] and CoDeepNeat [34] extend NEAT to mitigate its limitations when evolving the topologies and weights of the network. Besides, particle swarm optimization (PSO) has been analyzed to train RNNs instead of SGD [35], providing comparable results. ElSaid et al. [36] proposed the use of ant colony optimization (ACO) to improve LSTM RNNs by refining their cellular structure. Ororbia et al. [37] introduced the Evolutionary eXploration of Augmenting Memory Models (EXAMM), which is capable of evolving RNNs using a wide variety of memory structures. ElSaid et al. [38] introduced the Evolutionary eXploration of Augmenting LSTM Topologies (EXALT), a techniques for evolving RNNs, including epigenetic weight initialization and node-level mutation operations. Camero et al. [7] applied GA to search for the most efficient ones to improve the accuracy and the performance regarding the most commonly used RNNs configurations. In this case, the authors train the network using SGD to evaluate the performance of the configurations. As a conclusion of this literature review, it seems that the main difference between our approach and all the mentioned metaheuristics is that we propose to use the MRS instead of training each network/configuration. Thus, we expect to reduce the computational cost of the evaluation process, allowing the optimization algorithm to perform a larger number of iterations.

3. Random Error Sampling-based Neuroevolution

In this section, we introduce RESN, our proposal for RNN architecture optimization based on the MRS. First, (*i*) we state the architecture optimization problem and then, (*ii*) we present an evolutionary algorithm to perform such optimization.

3.1. Architecture Optimization

The optimization of the architecture of an artificial neural network consists of searching for an appropriate network structure (i.e., the architecture) and a set of weights [17]. However, in spite of this definition, it is rather common to arbitrarily define the architecture and then applied a learning rule (e.g., SGD) to optimize the set of weights [9]. Thus, we might say that the network is partially optimized or, in other words, we are not fully leveraging the computational model.

Usually, the RNN architecture optimization is stated as a minimization problem [9]. For example, we may define this problem as looking for an RNN architecture that minimizes the mean absolute error (MAE) of the predicted output (z_i) against the real one (y_i), subject to a minimum/maximum architecture (ARQ) definition (i.e., the number of hidden layers, the number of neurons per each layer, and the connecting edges), and to a minimum/maximum look back (LB). The training of the candidate solution is usually implied in this definition. Therefore, due to the intensive computations of the training, this optimization tends to be time demanding. Therefore, we propose to reformulate the optimization problem using the MRS.

We propose to optimize the architecture of an RNN by maximizing p_t, i.e., given an input X and an output Y, we propose to look for an RNN architecture that maximizes the estimated probability of finding a set of weights whose error is below a user-defined threshold (Algorithm 1). Equation 1 presents the referred problem.

$$\begin{aligned} &\text{maximize Heuristic} = p_t(X, Y) \\ &\text{subject to min\_ARQ} \leq ARQ \leq \text{max\_ARQ} \\ &\text{min\_LB} \leq LB \leq \text{max\_LB} \end{aligned} \tag{1}$$

3.2. Evolutionary Approach

To solve the RNN architecture optimization problem stated in Equation 1, we designed a *deep neuroevolutionary* algorithm based on the $(\mu + \lambda)$ EA [20]. Algorithm 2 presents a high-level view of our proposal.

Algorithm 2 Random Error Sampling-based Neuroevolution

```
1:  population ← Initialize(population_size)
2:  Evaluate(population)
3:  evaluations ← population_size
4:  while evaluations ≤ max_evaluations do
5:      offspring ← BinaryTournament(population, offspring_size)
6:      offspring ← CellMutation(offspring, cell_mut_p, max_step)
7:      offspring ← LayerMutation(offspring, layer_mut_p)
8:      Evaluate(offspring)
9:      population ← Best(population + offspring, population_size)
10:     evaluations ← evaluations + offspring_size
11:     SelfAdapting(layer_mut_p, max_step, cell_mut_p)
12: end while
13: solution ← Best(population, 1)
14: rnn_trained ← Train(solution, epochs)
15: return rnn_trained
```

Here, a solution represents an RNN architecture (i.e., ARQ in Equation 1), and it is encoded as a variable length integer vector, $sol =< s_0, s_1, ..., s_H >$, where s_0 is the LB $s_0 \in [\text{min\_LB}, \text{max\_LB}]$, and s_i $(i \in [1, \text{H}])$ corresponds to the number of LSTM cells in the i-th hidden layer. Thus, $s_i \in [\text{min\_NPL}, \text{max\_NPL}]$ and $H \in [\text{min\_HL}, \text{max\_HL}]$. Given this definition, the number of hidden layers is implicitly derived from the length of the solution. Then, the **population** is defined as a set of *population_size* solutions.

First, the **Initialize** function randomly creates a set of solutions (using a uniform distribution). Next, the **Evaluate** function computes *p_t* (Algorithm 1) for each solution. Then, the population is evolved until the termination criteria is met (i.e., the number of evaluations is greater than *max_evaluations*).

The evolutionary process is divided into selection, mutation, evaluation, replacement, and self-adjustment (Algorithm 2). First, (line 5) an **offspring** (of *offspring_size* solutions) is selected using a binary tournament from the actual population. Then, each solution in the offspring is mutated by a two step process. In the first step of the mutation (line 6, **CellMutation**), for every s_j $(j \in [0, H])$, with a probability *cell_mut_p*, a value in the range [min_step, max_step] (excluding zero) is added. Then, in the second step of the mutation (line 7, **LayerMutation**), independently, with a probability *layer_mut_p*, the layer s_i $(i \in [1, \text{H}])$ is cloned or removed (with the same probability), i.e., one layer is added or subtracted to the solution.

Once the mutation is done, the offspring is evaluated (**Evaluate**) using the MRS, and after, the best solutions from the population and the offspring are selected by the **Best** function, i.e., the population and the offspring are gathered together, sorted, and finally, the solutions that have a higher p_t give place to the new population (of *population_size* solutions).

The number of evaluations is increased by *offspring_size*. And finally, a **SelfAdapting** process takes place. In this process, if the new population is improving on average (i.e., the average p_t of the new population is greater than the former average), then, the *cell_mut_p*, *max_step*, and *layer_mut_p* parameters are multiplied by 1.5. Otherwise, these parameters are divided by 4. These numbers/values are taken from [39].

After the evolutionary process ends, the best solution (i.e., the solution with the greatest p_t) of the population is selected (line 13), and trained using a user-defined method. Without loss of generality, we defined to use Adam [40] optimizer to the train the final solution for a predefined number of *epochs*.

Finally, the algorithm returns an RNN that is optimized (structure and weights) to the given problem. Figure 1 depicts a high-level view of RESN.

It is quite interesting to notice that the **Evaluate** function may be changed seamlessly by any other fitness function, e.g., the MAE after training the network for a user-defined number of times. Accordingly, the **Best** function has to be modified to maximize or minimize the new objective function (fitness).

4. Experimental Setup

This section introduces the study performed. First, we present and justify the data sets. Second, we introduce the experiments carried out to test our approach.

4.1. Data Sets

To test our proposal and answer the research questions we selected four problems: the sine wave, the waste generation prediction problem [41], the coal-fired power plant flame intensity prediction problem [37], and the load forecast problem [42].

The *sine wave* problem consists of predicting the following value of the function, given the historical data. A sine wave is a periodic oscillation curve, that may be expressed as a function of time (t), where A is the peak amplitude, f is the frequency, and ϕ is the phase (Equation 2). Particularly, we used the sine wave described by $A = 1$, $f = 1$, and $\phi = 0$, in the range $t \in [0, 100]$ seconds (s), with ten samples per second. In spite of its simplicity, this problem is very useful because any periodic waveform can be approximated by adding sine waves and it is extensively used in the literature [10, 12, 43].

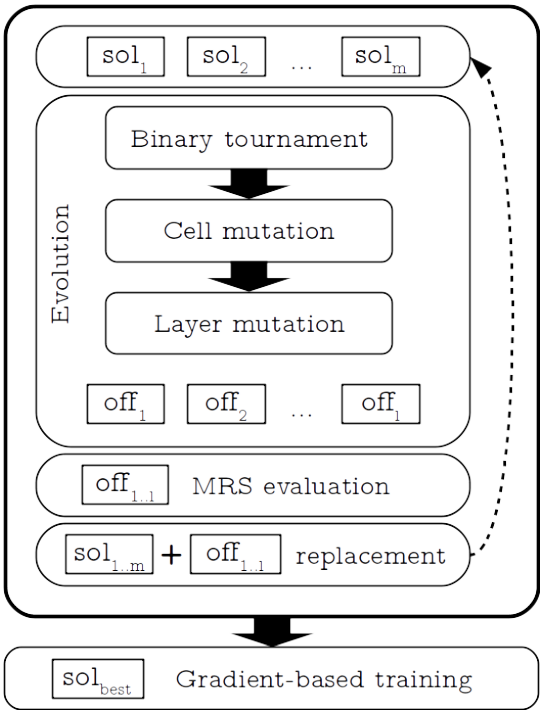

Figure 1: The global scheme of RESN

$$y(t) = A \cdot sin(2\pi \cdot f \cdot t + \phi) \quad (2)$$

The *waste generation prediction problem*, introduced by Ferrer and Alba [44], consists of predicting the filling level of 217 recycling bins located in the metropolitan area of a city in Spain. Each filling level is recorded daily for one year. Therefore, given the historical data (i.e., 217 input values per day), the problem is to predict the next day (i.e., the filling level of all bins).

It is important to notice that this problem was originally proposed to predict the filling level of each container individually using *Gaussian processes*, *linear regression*, and *SMReg* [44]. A later work (we will refer to this results as *Short training*) outperformed those results by predicting all containers at once using one single RNN [45].

The third validation case is the *coal-fired power plant flame intensity prediction problem*. It was introduced by Ororbia et al. [37], and consists of ten days of values recorded every minute for 12 burners. The original data set has been pre-normalized to the range [0,1]. The problem is to predict the main flame intensity given the historical data.

Finally, the *load forecast problem* [42], originally introduced by EUNITE network in the 2001 competition called "Electricity Load Forecast using Intelligent Adaptive Technologies"[1], consists of a mid-term load forecasting challenge. Particularly, the data set includes the electricity load demand of the Eastern Slovakian Electricity Corporation every half hour, from January 1, 1997, to January 31, 1999. As well as the temperature (daily mean), and the working calendar for the referred period. Then, using the data from 1997 to 1998, the challenge is to predict the maximum daily load of the January 1999 days.

[1] http://www.eunite.org/

4.2. Experiments

To answer the research questions we propose to implement RESN and benchmark it against state-of-the-art techniques to optimize the architecture of an RNN. Particularly, we propose the following four experiments.

4.2.1. E1: RESN vs. Gradient-based Architecture Optimization

To answer the question of whether a *hybrid* (MRS and gradient-based) architecture optimization technique can get the same error performance of a solely gradient-based one, we propose three tests. In all cases, based on the MRS original proposal [10], we set the MRS parameters according to Table 1. It is worth noticing that the parameters set in the referred table may influence the results. Nonetheless, we decided to rely on the values set in the original paper, as tuning them is not in the scope of this study.

Table 1: MRS parameters

Parameter	Value
MAX_SAMPLES	100
THRESHOLD	0.01

First, for the sine wave, we propose to (*E1.i*) run Algorithm 2 but using the *early* training results to evaluate the solutions, i.e., we propose to train each candidate architecture using Adam for a short time (one epoch), use the trained architecture to predict on the test data set, compute the MAE, and use that value as the heuristic of the optimization algorithm (Algorithm 2, line 8). For this test, we set the parameters according to Table 2. It is very important to notice that during the optimization *cell_mut_p, max_step, and layer_mut_p* values are self-adapted (Algorithm 2, line 11). Thus, their initial values are not critical [39].

Table 2: Parameters of E1.i: RESN vs. Gradient-based architecture optimization

Param	Value	Param	Value
cell_mut_p	0.2	min_LB	2
epochs	100	max_LB	30
max_step	5	min_NPL	1
layer_mut_p	0.2	max_NPL	100
population_size	10	min_HL	1
offspring_size	10	max_HL	3
max_eval	100		

Second, for the waste prediction problem, we propose to (*E1.ii*) benchmark our results against *Short training* [45]. Particularly, the authors proposed to optimize an RNN to the problem using an ES-based algorithm. More specifically, they trained each candidate solution using *Adam* [40] for a short time (ten epochs), and once the termination criteria were met, they trained the final solution for 1000 epochs. Therefore, in line with the parameters used in the referred paper, we set the parameters of our algorithm according to Table 3.

Table 3: Parameters of E1.ii: RESN vs. Gradient-based architecture optimization

Param	Value	Param	Value
cell_mut_p	0.2	min_LB	2
epochs	1000	max_LB	30
max_step	15	min_NPL	10
layer_mut_p	0.2	max_NPL	300
population_size	10	min_HL	1
offspring_size	10	max_HL	8
max_eval	100		

And third, as a sanity check, we propose to (*E1.iii*) optimize the architecture (constrained to the architecture search space defined in Table 3) using a *Random Search* algorithm.

Note that in all cases, we propose to split the data sets into training (64% of the data), validation (16%), and test (20%) data. Also, we set an early stop criteria for the *training* of the final solution (Algorithm 2, line 14), thus when the validation loss is below 1e-5 the training is stopped, and we propose to add a 0.5 dropout to that training.

4.2.2. E2: RESN vs. Neuroevolution

To answer the question of whether a *hybrid* architecture optimization technique can get a similar error performance of a non-training-based approach, we propose to (*E2*) benchmark our approach against EXALT [38], a state-of-the-art neuroevolutionary technique, on the coal-fired prediction problem. Also, we compare qualitative our results against EXAMM [37] (on the coal-fired prediction problem). Note that we did not propose a direct quantitative comparison because EXAMM evolves the architecture as well as the cell structures. Thus the problem being solved is slightly different. Moreover, in EXAMM the authors used a different experimental design (compared to [38]).

Particularly, we replicate the experimental design of EXALT [38]. Specifically in this experiment, we use K-fold cross validation, using each burner as a test case (i.e., $K = 12$). Additionally, accordingly to EXALT experimental design, we use *stochastic gradient descent* (instead of *Adam*) to train the networks (Algorithm 2, line 14) with a learning rate $\eta = 0.001$, utilizing Nesterov momentum with *mu*= 0.9, and without dropout. Also, we use gradient clipping when the norm of the gradient was above 1.0, and boosting when the norm of the gradient was below 0.05. Finally, we set RESN parameters according to Table 4. Note that the number of epochs of training of the final solution was defined considering the experimental design of EXALT [38].

Table 4: Parameters of E2: RESN vs. Neuroevolution

Param	Value	Param	Value
cell_mut_p	0.2	min_LB	2
epochs	1000	max_LB	30
max_step	5	min_NPL	1
layer_mut_p	0.2	max_NPL	100
population_size	10	min_HL	1
offspring_size	10	max_HL	8
max_eval	100		

4.2.3. E3: Optimization Time

As to answer RQ3, we propose to (*E3*) record the execution (run) time of E1 (because we do not have the execution times of E2 competitors), and use it to benchmark our proposal against the training-based architecture optimization techniques.

4.2.4. E4: RESN vs. Expert Design

To answer the question of whether RESN can improve the performance of an expert designed architecture/solution (RQ4), we propose to (E4) benchmark our proposal against the winning solution of the "Electricity Load Forecast using Intelligent Adaptive Technologies" [42], and with state-of-the-art approaches evaluated on this challenge [46].

In this case, we set RESN parameters to their *defaults* (Table 5). Also, we normalized the data to have a mean equal to zero and a standard deviation equal to one, according to the pre processing suggested by Lang et al. [46], and we set the activation function of the output layer to be `linear`.

Table 5: Parameters of E4: RESN vs. Expert design

Param	Value	Param	Value
cell_mut_p	0.2	min_LB	2
epochs	1000	max_LB	30
max_step	5	min_NPL	1
layer_mut_p	0.2	max_NPL	100
population_size	10	min_HL	1
offspring_size	10	max_HL	3
max_eval	100		

5. Experimental Results

To carry out the experiments defined in Section 4, we have implemented RESN in Python (code available in `https://github.com/acamero/dlopt`), using DLOPT [47], Keras [48], and Tensorflow [49]. Then, we run the experiments on the defined problems. Particularly, we repeated each experiment 30 independent times, and we computed the statistics of the error over the final solution.

First, we addressed *E1.i*. Table 6 summarizes the results of the *E1.i* experiment, where RESN stands for the optimization guided by the MRS heuristic and GDET for Adam training heuristc (defined in Section 4). The best mean and median are in bold font.

Table 6: E1.i results on the sine wave problem (MAE of the final solution)

	RESN	GDET
Mean	**0.105**	0.142
Median	**0.100**	0.149
Max	0.247	0.270
Min	0.063	0.054
Sd	0.035	0.051

Overall, the results of RESN exceed GDET, i.e., the optimized RNN obtained by RESN (guided by the MRS) has on

Table 7: E1.ii results. RNN optimization in the Waste generation prediction problem, a comparison between Short training and RESN

	Short training					*RESN*				
	MAE	No. LSTM	LB	No. HL	Time [min]	MAE	No. LSTM	LB	No. HL	Time [min]
Mean	**0.073**	451	6	5	97	0.079	793	17	3	**51**
Median	**0.073**	420	5	5	70	**0.073**	513	16	2	**45**
Max	0.076	1252	16	8	405	0.138	2038	30	8	103
Min	0.071	127	2	1	33	0.069	444	2	1	40
Sd	0.001	228	2	2	75	0.017	493	11	3	13

average a lower error than the ones optimized by GDET. Moreover, the Wilcoxon rank-sum test p-value is 0.001. Therefore, we can conclude that RESN is significantly better than GDET.

To continue with the validation of RESN, we executed the *E1.ii* experiment. Table 7 summarizes the results presented in [45] (columns under *Short training*), and our results (columns under RESN). In the table, *MAE* stands for the MAE of the final solution, *No. LSTM* is the number of LSTM in the network, *LB* corresponds to the look back, *No. HL* represents the number of hidden layers, and *Time* is the total time (i.e., the optimization process and the training of the final solution) in minutes. The best mean and median are in bold font.

In terms of the MAE, the results of both approaches (RESN and *short training*) are similar. Therefore, we performed a Wilcoxon rank-sum test to validate if there is a significant difference between them. Note that both approaches are stochastic and were executed 30 independent times for statistical soundness. The p-value of the test (comparing the MAE) is equal to 0.665, therefore there is no evidence that one algorithm *outperforms* the other. Furthermore, the median is the same in both cases.

Again, it is important to notice that Ferrer and Alba [44] originally proposed to predict the filling level of each container individually using *Gaussian processes*, *linear regression*, and *SMReg*. But later, *Short training* [45] outperformed those results. Then, our proposal beats all the techniques used in [44].

On the other hand, RESN (by using the MRS as the heuristic for optimizing the network) dramatically reduces the time needed to optimize the RNN configuration (RQ3). On average, the time has been cut in half (nearly one hour difference). Again, notice that Table 7 presents the time in minutes.

We run the *E1.iii* experiment. On average, the MAE of the solution found using the random search was equal to 0.091 for the waste prediction problem, and the training time of each network was 25 minutes. Therefore, evaluating 100 networks (*max_eval*) took nearly 50x (on average) the time needed to optimize the network using RESN. We also performed a Wilcoxon rank-sum test to compare the Random search against RESN and Short training. The p-values are 0.017 and 0.002 respectively. Therefore, we concluded that the Random search does not perform as well as the other techniques. Nonetheless, it is quite interesting that a simple random search could give such good results (MAE). However, its main drawback is the time needed to get a *competitive* solution.

Later, we run the E2 experiment. Table 8 presents the mean square error (MSE) of the solution obtained by RESN in the coal-fire power plan problem, as well as the results presented in

Table 8: E2 results (MSE). RESN vs EXALT in the coal-fire power plant problem

Fold	EXALT	RESN
0	0.028749	**0.001541**
1	0.031769	**0.006536**
2	0.023095	**0.003821**
3	0.019229	**0.000570**
4	0.023170	**0.003336**
5	0.036091	**0.000617**
6	**0.012879**	0.017061
7	0.019358	**0.004032**
8	0.018151	**0.001912**
9	0.019475	**0.013996**
10	0.030016	**0.006120**
11	0.031207	**0.002942**
Average	0.024432	**0.005208**

EXALT [38]. The results show that RESN improves the results of EXALT, with a difference of one order of magnitude. The best result for each fold and for the average is in bold font. In this case, the Wilcoxon rank-sum test p-value is 2.958e-06. Despite the small decimal figures involved, our technique has shown to be ten times more accurate than the state-of-the-art, and the statistical tests confirm that this difference is meaningful.

Then, we processed the results of EXAMM [37]. Particularly, we computed the average MSE for RNNs Evolved With Individual Memory Cells and RNNs Evolved With All Memory Type (Table 1 in the cited article) and for RNNs Evolved With Simple Neurons and Memory Cells (Table 2 in the cited article). The values are 0.001690 and 0.001601, respectively. Although the experiments are not comparable (i.e., EXAMM uses a different experimental design, but the same data set), it is quite interesting to notice that RESN achieves a result (0.005208) that is in the same order of magnitude that EXAMM, but with a much simpler approach, i.e., we only use fully connected stacked LSTM layers, instead of multiple types of cells.

Finally, we run the E4 experiment. In this case, we reported the mean absolute percentage error (MAPE), as it is the metric reported in the benchmarked studies.The summarized results are presented in Table 9. The column SVM corresponds to the results presented by Chen et al. [42], the winner of the "Electricity Load Forecast using Intelligent Adaptive Technologies" competition organized by EUNITE. In the referred study, the authors proposed a support vector machine approach to predict the load. It is important to notice that they also studied in deep

Table 9: E4 results (MAPE). RESN vs Expert design

	SVM	BP	RBF	SVR	NNRW	KNNRW	WKNNRW	RESN
Mean	2.879	NA	NA	NA	NA	NA	NA	2.281
Median	2.945	NA	NA	NA	NA	NA	NA	2.242
Max	3.480	NA	NA	NA	NA	NA	NA	3.273
Min	1.950	1.451	1.481	1.446	1.438	1.348	1.323	1.370
Std	0.004	NA	NA	NA	NA	NA	NA	0.414

the data set, and proposed several preprocessings to the prepare the data for the specific task.

On the other hand, the columns BP, RBF, SVR, NNRW, KNNRW, and WKNNRW correspond to the results presented by Lang et al. [46]. In this study, the authors proposed several approaches to tackle the load forecast problem. Particularly, they proposed to use BP (backpropagation neural network), RBF (radial basis function network), SVR (support vector regression), NNRW (neural network with random weights), KNNRW (neural network with random weights and kernels), and WKNNRW (weighted neural network with random weights and kernels). As a remark, the authors did not report statistical results, thus we added a NA (not available) to the missing data.

Overall, the performance of RESN is comparable to a human expert, meaning that the error of the best solution found is as good as the best solution proposed by the experts. We acknowledge that the techniques are not the same, thus there may be a bias in this benchmark. Nonetheless, the comparison is *fair* in terms that all the considered approaches were specifically tailored for the load forecast problem, and that all of them were tuned manually with expert domain/technique knowledge.

As a summary, RESN has a similar (or better) error performance to training-based RNN optimization techniques (RQ1) and to neuroevolutionary approaches (RQ2) but considerably reduces the computational time (RQ3). Moreover, the performance of RESN is as good as the best solution proposed by an expert (RQ4). Hence, we offer a competitive alternative to architecture optimization that does not rely on training but on the MRS, and that does not require expert domain/technique knowledge to tailor a solution a specific problem.

6. Conclusions and Future Work

In this work, we present RESN, an EA to optimize the architecture of an RNN that uses the MRS as a search heuristic. We have evaluated our proposal on four prediction problems and compared our results against training-based architecture optimization techniques, neuroevolutionary approaches, and human expert designed approaches.

The results show that RESN is as good as a training-based architecture optimization technique in terms of the error, i.e., the prediction error of our solutions are similar to the ones of the architectures optimized using training-based architecture optimization techniques.

When comparing RESN against *pure* neuroevolutionary techniques, the results show that RESN achieves state-of-the-art performance. Moreover, the evidence presented in this work shows that our approach reduces by half the time needed to optimize the architecture of an RNN compared to a training-based architecture optimization technique (on average, compared to short training results, we reduce the time from 97 to 51 minutes in the waste generation prediction problem, and RESN is 50x faster than training each candidate architecture). Therefore, in the future, we could optimize much larger networks in the time that existing algorithms need for smaller cases.

Moreover, the results show that RESN is a competitive replacement to human expert design solutions, with the add-on that no domain specific knowledge is required to tailor the RNN to the problem.

According to these results, we conclude that RESN provides a competitive alternative for RNN optimization. Overall, the results suggest that the MRS is a promising method to compare RNN architectures and that it is a very useful heuristic for architecture optimization.

As future work, we propose to extend our proposal (and the MRS) to other problem classes, e.g., classification, clustering, among others. Moreover, we will analyze the use of other metaheuristics, such as, a GA with specific operators, that will allow to improve the search of network architectures. On the other hand, we envision the importance of studying thoroughly the configuration of the MRS parameters, including using a different PDF, training algorithm, among others.

Acknowledgments

This research was partially funded by Universidad de Málaga, Andalucía Tech, Consejería de Economía y Conocimiento de la Junta de Andaluía, Ministerio de Economía, Industria y Competitividad, Gobierno de España, 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). European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 799078. Helmholtz Association's Initiative and Networking Fund (INF) under the Helmholtz AI platform grant agreement (ID ZT-I-PF-5-1).

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, nature 521 (2015) 436.

[2] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, C. I. Sánchez, A survey on deep learning in medical image analysis, Medical image analysis 42 (2017) 60–88.

[3] S. Min, B. Lee, S. Yoon, Deep learning in bioinformatics, Briefings in bioinformatics 18 (2017) 851–869.

[4] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE tran on neural networks 5 (1994) 157–166.

[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, ICML 13, JMLR.org, 2013, pp. III–1310–III–1318.

[6] 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.

[7] 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.

[8] 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, ICML'15, JMLR.org, 2015, pp. 2342–2350.

[9] V. K. Ojha, A. Abraham, V. Snášel, Metaheuristic design of feedforward neural networks: A review of two decades of research, Engineering Applications of Artificial Intelligence 60 (2017) 97–116.

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

[11] 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, IJCAI'15, AAAI Press, 2015, pp. 3460–3468.

[12] A. Camero, J. Toutouh, E. Alba, Comparing deep recurrent networks based on the mae random sampling, a first approach, in: Conference of the Spanish Association for Artificial Intelligence, Springer, 2018, pp. 24–33.

[13] Z. C. Lipton, J. Berkowitz, C. Elkan, A critical review of recurrent neural networks for sequence learning, arXiv preprint arXiv:1506.00019 (2015).

[14] J. F. Kolen, S. C. Kremer, Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, Wiley-IEEE Press, 2001, pp. 464–479.

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

[16] 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), 2017, pp. 1–7.

[17] S. Haykin, Neural networks and learning machines, volume 3, Pearson Upper Saddle River, NJ, USA:, 2009.

[18] S. Albelwi, A. Mahmood, A framework for designing the architectures of deep convolutional neural networks, Entropy 19 (2017) 242.

[19] S. C. Smithson, G. Yang, W. J. Gross, B. H. Meyer, Neural networks designing neural networks: multi-objective hyper-parameter optimization, in: Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on, IEEE, 2016, pp. 1–8.

[20] T. Back, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford university press, 1996.

[21] E. Alba, J. Aldana, J. M. Troya, Full automatic ann design: A genetic approach, in: International Workshop on Artificial Neural Networks, Springer, 1993, pp. 399–404.

[22] E. Alba, R. Martí, Metaheuristic procedures for training neural networks, volume 35, Springer Science & Business Media, 2006.

[23] Y. Zhining, P. Yunming, The genetic convolutional neural network model based on random sample, International Journal of u-and e-Service, Science and Technology 8 (2015) 317–326.

[24] G. Rosa, J. Papa, A. Marana, W. Scheirer, D. Cox, Fine-tuning convolutional neural networks using harmony search, in: A. Pardo, J. Kittler (Eds.), Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer Intl Pub, Cham, 2015, pp. 683–690.

[25] L. R. Rere, M. I. Fanany, A. M. Arymurthy, Simulated annealing algorithm for deep learning, Procedia Computer Science 72 (2015) 137 – 144. The Third Information Systems International Conference 2015.

[26] L. Rere, M. I. Fanany, A. M. Arymurthy, Metaheuristic algorithms for convolution neural network, Computational intelligence and neuroscience 2016 (2016).

[27] C. Wang, C. Xu, X. Yao, D. Tao, Evolutionary generative adversarial networks, IEEE Transactions on Evolutionary Computation 23 (2019) 921–934.

[28] T. Schmiedlechner, I. N. Z. Yong, A. Al-Dujaili, E. Hemberg, U.-M. O'Reilly, Lipizzaner: A System That Scales Robust Generative Adversarial Network Training, in: the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018) Workshop on Systems for ML and Open Source Software, 2018.

[29] J. Toutouh, E. Hemberg, U.-M. O'Reilly, Spatial evolutionary generative adversarial networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 472–480. URL: https://doi.org/10.1145/3321707.3321860. doi:10.1145/3321707.3321860.

[30] K. O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary computation 10 (2002) 99–127.

[31] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks, Journal of machine learning research 10 (2009) 1–40.

[32] 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.

[33] 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.

[34] J. Liang, E. Meyerson, R. Miikkulainen, Evolutionary architecture search for deep multitask networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, ACM, New York, NY, USA, 2018, pp. 466–473. URL: http://doi.acm.org/10.1145/3205455.3205489. doi:10.1145/3205455.3205489.

[35] A. M. Ibrahim, N. H. El-Amary, Particle swarm optimization trained recurrent neural network for voltage instability prediction, Journal of Electrical Systems and Information Technology 5 (2018) 216 – 228.

[36] 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, GECCO '18, ACM, New York, NY, USA, 2018, pp. 13–20. URL: http://doi.acm.org/10.1145/3205455.3205637. doi:10.1145/3205455.3205637.

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

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

[39] C. Doerr, Non-static parameter choices in evolutionary computation, in: Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings, ACM, 2017. doi:10.1145/3067695.3067707.

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

[41] J. Ferrer, E. Alba, Bin-ct: Urban waste collection based in predicting the container fill level, arXiv preprint arXiv:1807.01603 (2018).

[42] B.-J. Chen, M.-W. Chang, et al., Load forecasting using support vector machines: A study on eunite competition 2001, IEEE tran on power systems 19 (2004) 1821–1830.

[43] R. N. Bracewell, R. N. Bracewell, The Fourier transform and its applications, volume 31999, McGraw-Hill New York, 1986.

[44] 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.

[45] A. Camero, J. Toutouh, J. Ferrer, E. Alba, Waste generation prediction in smart cities through deep neuroevolution, in: Ibero-American Congress on Information Management and Big Data, Springer, 2018, pp. 192–204.

[46] 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 Computing (2018) 1–9.

[47] A. Camero, J. Toutouh, E. Alba, Dlopt: deep learning optimization library, arXiv preprint arXiv:1807.03523 (2018).

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

[49] 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, volume 16, 2016, pp. 265–283.