2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

An Effective Criterion for Pruning Reservoir's Connections in Echo State Networks

Simone Scardapane, Gabriele Nocco, Danilo Comminiello, Michele Scarpiniti and Aurelio Uncini

***Abstract*— Echo State Networks (ESNs) were introduced to simplify the design and training of Recurrent Neural Networks (RNNs), by explicitly subdividing the recurrent part of the network, the *reservoir*, from the non-recurrent part. A standard practice in this context is the random initialization of the reservoir, subject to few loose constraints. Although this results in a simple-to-solve optimization problem, it is in general suboptimal, and several additional criteria have been devised to improve its design. In this paper we provide an effective algorithm for removing redundant connections inside the reservoir during training. The algorithm is based on the correlation of the states of the nodes, hence it depends only on the input signal, it is efficient to implement, and it is also local. By applying it, we can obtain an optimally sparse reservoir in a robust way. We present the performance of our algorithm on two synthetic datasets, which show its effectiveness in terms of better generalization and lower computational complexity of the resulting ESN. This behavior is also investigated for increasing levels of memory and non-linearity required by the task.**

I. Introduction

RESERVOIR Computing (RC) is a well-established paradigm for designing and training recurrent neural networks (RNNs) [1]. It unifies the guiding principles of earlier models such as Echo State Networks (ESNs) proposed in 2001 by Jaeger [2] and Liquid State Machines (LSMs) proposed in the same year by Maass et al. [3]. The key idea is to consider the overall network as composed of two components: a recurrent *reservoir* and a static *readout*. The reservoir is driven by the input signal to generate a rich set of dynamic features, while the readout is trained using standard machine learning methods on top of the features extracted from the reservoir.

Since the two parts perform a different role, their design can vary dramatically. In ESNs, in particular, the reservoir is generally built from standard non-linear neurons, whose connectivity is randomly created at the beginning of the training process in the form of an $N \times N$ matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$, where N is the size of the reservoir itself. The readout is then constructed as a linear layer trained using ridge regression (in the batch version) or linear adaptive filters (in the online case). To ensure stability, the reservoir has to satisfy the *echo state property*, i.e., the effect of an input on the state of the reservoir should vanish after a certain amount of time [1]. In this case, it is said that the network has a *fading memory*. Although there are several criteria for ensuring this property, the most common is to rescale $\mathbf{W}$ such that its *spectral radius* $\rho(\mathbf{W})$[1] is less than unity. Jaeger [2] introduced this as an heuristic criterion, and showed it to be necessary when using specific non-linearities in the reservoir neurons in the case of zero input[2]. The spectral radius has in fact a fundamental role in determining the dynamic regime of the network [4], balancing between the memory capacity allowed to the reservoir and the level of non-linearity introduced in its computations.

The random initialization of the reservoir is at the hearth of the success of ESNs, since it transforms the complex task of training an RNN into a much simpler linear regression. Jaeger identified a "good" reservoir as having, in addition to the random connectivity, (i) a large size and (ii) sparsity of the connections [1]. Informally, this last property translates into a range of dynamical features as diversified as possible, hence reducing redundancy and possibly improving generalization of the system. Jaeger proposed to achieve sparsity by choosing a degree of connectivity d, and subsequently generating only d percent of the connections inside the reservoir. However, choosing an optimal d introduces an additional parameter into the design of the ESN, which already involves several other conflicting factors. Moreover, the random choice of the connections to be generated does not result in general in a significant amount of improvement. These factors probably explain the wide diffusion of considering fully-connected reservoirs, e.g. in [6], [7].

Separately from this, it is also clear that the random initialization of the weights is sub-optimal in the context of solving a specific problem, as detailed in the following quote of Lukoevicius and Jaeger [1]: "*it seems obvious that, when addressing a specific modeling task, a specific reservoir design that is adapted to the task will lead to better results than a naive random creation*". For this reason, a wide range of additional strategies have been devised to improve over it, both supervised and unsupervised (for a review up to 2009, see [1, Section 5]).

In this paper we are interested in addressing both problems by investigating a relatively unexplored area in ESN literature: online pruning of the reservoir's connections during training. The efficacy of pruning on large RNNs is known since the seminal work of [8], and from the previous discussion an automatic pruning criterion in the context of ESNs would have the following advantages: (i) it would allow to obtain optimally sparse reservoir without having to specify

Authors are with the Department of Information Engineering, Electronics and Telecommunications (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184, Rome. Email: {simone.scardapane, michele.scarpiniti, danilo.comminiello, aurelio.uncini }@uniroma1.it, gabriele.nocco@codin.it.

[1]The spectral radius of a matrix $\mathbf{A}$ is defined as $\rho(\mathbf{A}) = \max_i(|\lambda_i|)$, where λ_i are the eigenvalues of $\mathbf{A}$.

[2]Recent works have derived more formal bounds involving the spectral radius itself, see [5].

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

the sparsity factor *a priori*; (ii) this in turn would decrease the computational cost of updating the reservoir, which is fundamental for hardware implementations [9]; (iii) it would help in generalizing better and obtaining less redundant features, since the pruning process can also be guided by the actual input signal. Additionally, pruning has also an interesting biological foundation investigated under the field of *neuroplasticity* [10]. In fact, in the human brain, synapses grow at a constant rate until a peak is reached around the age of 2-3 years, and are subsequently removed until their density stabilizes in adulthood [10]. Although seemingly wasteful, this process is part of an essential trade-off between the required plasticity and the energy requirements of the brain during its development. Under a very broad metaphor, we can say that an ESN with full connectivity and randomly generated synapses is still in its "infancy". In this sense, it would be beneficial to develop simple and computationally efficient strategies for deleting the least useful connections in an online manner, as is happening in the brain itself. In fact, some of the earliest approaches to pruning of a fully connected RNN originated from a biological analogy [11].

Up to now, the only work that has dealt explicitly with the idea of pruning in ESNs is the research by Dutoit et al. [12], that limited their investigation to the pruning of links from the reservoir to the readout. This has the advantage of transforming the problem into the well-studied task of sparsification of a linear model, for which very efficient algorithms are known [13]. Moreover, the sparsification has also a clear theoretical justification in that it is known to have a regularizing effect on the solution[3]. Pruning inside the reservoir, although potentially more beneficial due to the typical size of reservoirs, is instead more subtle. In particular, two problems arise. Firstly, setting to zero an entry of a matrix can increase its spectral radius (we show a simple example in Section IV). Hence, this can potentially destroy the echo state property[4]. Additionally, the dynamical nature of the reservoir makes sensitivity analysis based on error propagation and on gradient calculation hard, as is the case in standard RNN training.

Since the main aim of this paper is detailing a working algorithm, a theoretical analysis of the former point goes beyond its scope. Nonetheless, for completeness we present a brief discussion of it in Section IV, by considering a random deletion of synapses from a reservoir. Although our reasoning leads us to expect, in average, a decrease in spectral radius, our experiments show that a realistic pruning criterion can have a more complex behavior and can result in an automatic selection of the spectral radius itself. We expect to present an analysis of this point in a future work.

In Section III we propose our pruning strategy, which does not require Hebbian plasticity of the weights, nor the computation of the error gradients. In particular, we define the *significance* of a synapse, and thus its relative importance, in terms of the correlation between the states of its input and output neurons. At fixed intervals we perform a pruning operation, where each synapse has a probability of being removed which is inversely proportional to its significance. Since we require that this probability is decreasing over time, to define it we adopt a strategy very similar to the one used in the standard Simulated Annealing algorithm [16]. The probability evolution over time follows an exponential profile, whose exponent is guaranteed to decrease by the inclusion of an additional parameter, known as *temperature*. Its specification allows the user to have full control over the pruning profile, although retaining a simple modality of configuration, as we show in our experiments.

[3]Butcher et al. have investigated similar ideas [14], although in relation to a customized architecture built on an ESN.

[4]For the interested reader, we note that the interplay between a network's dynamical regime and synaptic pruning has been partly explored in the context of random neural networks [15].

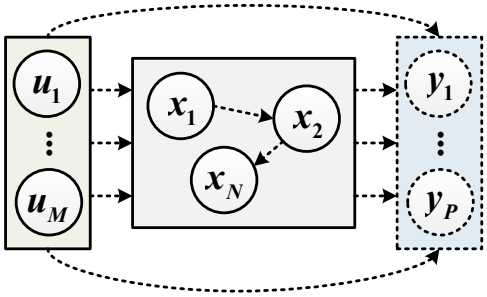

Fig. 1. Schema of the ESN used in our work. Fixed and trainable connections are represented with dashed and solid lines respectively.

The other Sections of the paper are organized as follows. We present the basic notation of the ESN used in our work in Section II. After detailing our pruning strategy, we perform a large set of simulations in Section V. A first experiment in Section V-A shows the efficacy of the algorithm, along with its simplicity in configuration. The second set of experiments in Section V-B investigate its behavior for increasing levels of memory and non-linearity required by the task. We conclude with some final remarks in Section VI.

NOTATION

In the rest of the paper, we adopt the following conventions. Vectors and matrices are denoted by boldface letters, which are in lowercase and uppercase respectively. All vectors are column vectors. The notation a_i denotes the i-th element of vector $\mathbf{a}$, and A_{ij} denotes the element in the i-th row and j-th column of matrix $\mathbf{A}$. Finally, $\mathbf{a}(n)$ denotes the value of a time-dependent vector at time-instant n.

II. ECHO STATE NETWORKS

We detail here the notation for a basic form of ESN, whose schematic representation is shown in Fig. 1. We denote by M the dimensionality of the input vector, by N the number of neurons in the reservoir, and by P the dimensionality of the desired output vector. At time n the reservoir is fed with input $\mathbf{u}(n)$ and its state is updated according to:

$$\mathbf{x}(n) = f(\mathbf{W}_r^r \mathbf{x}(n-1) + \mathbf{W}_i^r \mathbf{u}(n)) \qquad (1)$$

where $\mathbf{W}_r^r$ is the $N \times N$ matrix connecting the reservoir to itself, $\mathbf{W}_i^r$ is the $N \times M$ matrix connecting the input to the reservoir, and $f(\cdot)$ is the activation function of the neurons inside the reservoir. Both matrices are randomly generated and then rescaled according to the design requirements. Typically, $\mathbf{W}_r^r$ is rescaled so that its spectral radius $\rho(\mathbf{W}_r^r)$ is less than unity. Note that a bias is easily considered by inserting a constant unitary input in addition to $\mathbf{u}(n)$. Moreover, several forms of additional memory can be embedded in equation (1), as detailed in [17]. The output of the network is then given by:

$$\mathbf{y}(n) = g(\mathbf{W}_r^o \mathbf{x}(n) + \mathbf{W}_i^o \mathbf{u}(n)) \tag{2}$$

where $\mathbf{W}_r^o$ is the $P \times N$ matrix connecting the reservoir to the output, $\mathbf{W}_i^o$ the $P \times M$ matrix connecting the input to the output layer, and $g(\cdot)$ is the output activation function (considered as the identity function in our work, hence $g(\mathbf{s}) = \mathbf{s}$). Equations (1) and (2) can be extended to take into consideration additional connections, such as output-to-reservoir or feedback connections from the output.

The connections to the reservoir are randomly generated, hence the only free parameters to learn are in the matrices $\mathbf{W}_r^o$ and $\mathbf{W}_i^o$. Suppose the network is fed with an input sequence of length S $\{\mathbf{u}(1), \dots, \mathbf{u}(S)\}$, and produces the states $\{\mathbf{x}(1), \dots, \mathbf{x}(S)\}$. Denote by $\mathbf{Y} = [\mathbf{d}(1), \dots, \mathbf{d}(S)]$ the concatenation of all the desired outputs, by $\mathbf{s}(n)$ the "extended" state $\mathbf{s}(n) = \left[\mathbf{u}(n)^T \, \mathbf{x}(n)^T\right]^T$ and by $\mathbf{A}$ the concatenation of all such states $A = [\mathbf{s}(1), \dots, \mathbf{s}(S)]$. The weight matrix of minimum error and minimum norm is then given by:

$$\mathbf{W}_{\text{opt}} = \mathbf{A}^{\dagger}\mathbf{Y} \tag{3}$$

where $(\cdot)^{\dagger}$ denotes the operation of pseudo-inverting a matrix, and $\mathbf{W}_{\text{opt}}$ is the concatenation of the optimals $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$. Equation (3) can also be augmented by a regularization term or, as in our case, computed online using a linear adaptive filter [18]. First-order filters such as the *Least Mean-Square* are found in practice to have a poor performance, particularly in terms of convergence speed [1], due to the eigenvalue spread of the cross-correlation matrix of the states $\mathbf{x}(n)$. For this, second-order algorithms such as the *Recursive Least-Square* (RLS) are generally used. For completeness, we detail the RLS algorithm used in our experiments in the appendix.

We conclude this section with two final remarks. First, we note that the initial values produced by the network in a sequence are sometimes discarded due to their transient state, and are denoted as *dropout* elements. Secondly, multiple sequences in input can be easily handled by concatenating the resulting matrices.

Algorithm 1: Pseudo-code for a single update step of the pruned ESN, using the exponential cooling profile and the online update with RLS.

Data: Input signal $\mathbf{x}(n)$, desired output $d(n)$.
Compute new state of ESN with Eq. (1)
Update significance in Eq. (4)
if $(n/Q) - 1 = 0$ **then**
  Compute probabilities p_{ij} according to Eq. (5)
  Prune each synapse with probability p_{ij}
end
Apply RLS update step

III. Pruning by Node Correlation

In this section we detail a simple pruning strategy that does not require the computation of the error gradients[5]. It can be seen as an "hard thresholded" version of the standard Hebbian rule, and it is also loosely inspired to some biologically existing processes in the brain [10]. The main idea of Hebbian learning is that each connection in a neural network, a synapse, has a relative importance depending on how strongly correlated are the states of its corresponding neurons. This correlation can be easily computed in real-time, and it is used in Hebbian rules to adapt the synapse's weights. In our case, we define the *significance* of a synapse at time instant n, and thus its importance, as:

$$s_{ij}(n) = \frac{1}{T} \sum_{z=n-T}^{n} \frac{(x_i(z-1) - \hat{\mu}_x)(x_j(z) - \hat{\mu}_x)}{\hat{\sigma}_x^2} \tag{4}$$

where T is a time-interval chosen *a priori*, and $\hat{\mu}_x$ and $\hat{\sigma}_x$ are the empirical estimations of the mean and standard deviation of the neuron states. For simplicity, we suppose these two quantities to be equal for all neurons. Relaxing this constraint did not provide significant increases in performance in our simulations. Eq. (4) is similar in spirit to the classical *Pearson correlation coefficient* used in statistics for defining the level of linear correlation between random variables. We use $s_{ij}(n)$ to define a probability that a synapse is removed as:

$$p_{ij}(n) = \exp\left\{-\frac{|s_{ij}(n)|}{t(n)}\right\} \tag{5}$$

where $t(\cdot)$ is a positive, monotonically decreasing function of n, to take into account the fact that the probability of removing a synapse should be maximal in the beginning of learning (what is called a *critical phase* in RNN literature [19]) and decreases afterwards. Since this is inspired from the Simulated Annealing algorithm [16], we adopt the corresponding terminology and call $t(n)$ the *temperature* of the system. Every Q time instants, we prune each synapse with a

[5]In the following, we restrict ourselves to the deletion of redundant connections inside the reservoir, although everything we say extends naturally to the pruning of other connections in the network, such as input-to-reservoir links.

probability given by (5). In our experiments, the exponential profile for $t(n)$ worked very well:

$$t(n) = \alpha^{(n/Q)-1} t_0 \tag{6}$$

where t_0 is the initial temperature, to be chosen *a priori*, and α is called the *scaling factor*. Concretely, the temperature is scaled by a factor α at every "pruning step", given by $(n/Q) - 1$. Using this profile, it is intuitively clear that the scaling factor α will balance between number of connections and final testing accuracy. In particular, we expect a sufficient amount of pruning leading to better generalization, and an excessive amount of pruning leading to a deterioration of the performance. We note that many other choices are possible for the cooling strategy [20]. The pseudocode for a single update step of the ESN with our pruning algorithm is summarized in Algorithm 1.

Our strategy is partly task-dependent, in that it depends on the input elements of the sequence, although not directly on the output values (since we do not consider in this work feedback connections from the output layer)[6]. In the classification of [1], it can be considered as an "unsupervised local method". This property is important since the reservoir can in some situations be shared between multiple tasks. Moreover, it allows its use with both batch and online learning algorithms (as the ones described in Section II). However, since the strategy is online in nature, in our experiments we always preferred an online algorithm for training, although very similar results can be obtained in the former case. We also note that our pruning algorithm does not change the computational complexity of training an ESN, since the summation term in Eq. (4) can be computed for all synapses with a single outer product.

IV. Pruning and the Echo State Property

Pruning a single connection in the reservoir is equivalent to setting to zero an entry in $\mathbf{W}_r^r$. This setting has already been partly explored in the context of spectral graph theory [21]. Differently from it, however, the reservoir's matrix is not required to be non-negative, hence this operation can potentially increase its spectral radius. In some specific situations, this can be shown to disrupt the echo state property [1], [5]. As a simple example, consider the 2×2 matrix given by:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

which has $\rho(\mathbf{A}) = \sqrt{2}$. Deleting the element A_{22} leads to a new matrix $\mathbf{A}'$ with increased spectral radius $\rho(\mathbf{A}') = \frac{1+\sqrt{5}}{2}$.

Still, we should expect that a series of pruning operations on the reservoir leads, in average, to a decrease of its spectral radius. To understand this, consider the following informal reasoning. The sequence of pruning operations defines a sequence of matrices converging to the zero matrix $\mathbf{0}$ (the one in which all connections have been removed). Since $\rho(\mathbf{0}) = 0$, by the continuity of the spectral radius with respect to the matrix weights, and since we are generating matrices that are less and less dense, we expect the spectral radius of the series can decrease in average, despite each pruning operation has a probability of increasing it.

[6]Although the actual output can influence the choice of the scaling factor, which in turn influences the final number of connections.

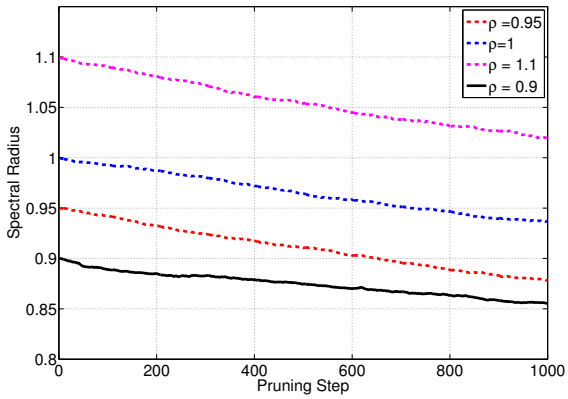

Fig. 2. Average behaviour of the spectral radius under pruning.

To check this intuition, we performed the following experiment. We generated a 100×100 matrix with weights taken from a normal Gaussian distribution, and scaled it so that its spectral radius matches a desired value, as it is a standard practice in ESN initialization. Then, we iteratively set to 0 a randomly chosen element of the matrix, until 10% of the connections were removed. The results of this experiment, averaged over 100 runs, are presented in Fig. 2 for 4 different initial spectral radii. The decreasing trend of the spectral radius is clearly visible for each initial choice. Hence, in practice there only exists a slight probability that the network looses stability after a random pruning operation whenever it is started very close to the so-called "edge of stability" [1], the boundary between stable and unstable regime.

However, as we show in Section V-B, the actual behavior of a pruning algorithm as the one we detailed in Section III can be more complex. In particular, in some situations in our experiments the spectral radius increases after removing a given number of connections, and it stabilizes close to unity. We expect to present in a future work the theoretical aspects that connect the spectral radius of the matrix with the evolution of the pruning process. Although the reservoir never lost stability in our experiments, at this point we cannot present a formal result for ensuring it, neither in the random case nor in the case of our specific algorithm, unless tracking the spectral radius in the initial stages of pruning. Note that this stability problem is partially avoided if we drop the constraint on online learning, since in this case it is possible to check the final spectral radius simply prior to the final training process.

V. Experimental Results

A. 10-th Order NARMA series

We start by comparing an unpruned ESN with a pruned version on the classical 10-th order NARMA system shown

TABLE I

EXPERIMENTAL RESULTS FOR THE 10-TH ORDER NARMA SERIES (MEAN AND STANDARD DEVIATION).

Architecture	MSE
Standard ESN	0.00179 ± 0.0001
Pruned ESN	0.00177 ± 0.0001

in [18]. Despite our pruning procedure does not results in a significant decrease in error in this case, we decided to include it in this section because it represents a standard experiment in this context, and it allows us to investigate the general behavior of the algorithm, which was found consistent on all the other experiments we performed. The next subsection will then show how the algorithm can eventually enhance (even in a significant way) the performance of the network. Following the original setup, we used a reservoir of 100 elements with $\tanh(\cdot)$ nonlinearities in the nodes. The input-to-reservoir matrix is initialized with full connectivity and weights extracted uniformly from the set $\{-0.1, 0.1\}$. The reservoir weights are extracted from a normal Gaussian distribution, and then $\mathbf{W}_r^r$ is rescaled to have a spectral radius of 0.9. A standard RLS filter is used for training with forgetting factor 0.995. Some additional Gaussian noise with variance 0.001 is injected into the state update to ensure numerical stability [18]. Using these settings, we found that the RLS has an equivalent performance with respect to a regularized offline training using ridge regression. We generated 20 sequences of 1500 elements each, and tested the system using a 10-fold cross validation over the sequences (i.e., each training fold contains 18 sequences). We prune the network every 100 time instants, and for simplicity we set $T = Q = 100$. The parameter α in (6) is found by performing an additional 3-fold cross validation over the training sequences on the following range: $[0.1, \ldots, 0.9]$, while the initial temperature is fixed at $t_0 = 0.3$. This last choice has a relatively small influence on the results, and very similar performance were found for other values of t_0, and other sizes of the reservoir. The mean squared error averaged over the folds is shown in Table I, along with ± 1 standard deviation.

As hinted previously, we see that the introduction of the pruning operation does not affect performance, although it does not improve it either in this experiment. However, we will use it to show some characteristics of his behavior. First of all, we note that the resulting reservoirs using pruning can be significantly smaller in terms of connections than the original version, which is a highly desirable characteristic for computational requirements and possible hardware implementations. In Fig. 3 we show the decrease in the number of connections, averaged over the 10 folds. We see that, with respect to the full reservoir, we are able to prune more than half of the original connections (the average final number of connections being ≈ 4800). The trend in Fig. 3 reflects

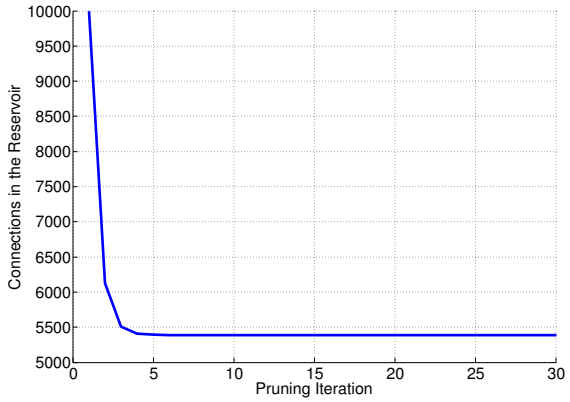

Fig. 3. Evolution of the reservoir density after each pruning operation.

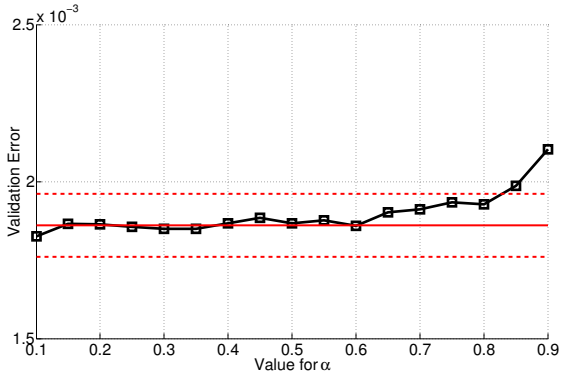

Fig. 4. Validation error for each choice of the scaling factor.

closely the choice of the temperature function, with a high number of connections deleted in the first iteration, a fewer number deleted in the following iterations, and a stabilization around iteration 6, i.e., around time instant $n = 600$, when the temperature is very close to 0. Moreover, the pruning operation introduces only a small perturbation in the spectral radius of the final network, which was found to be around 0.87. This behavior is expected since the original value of 0.9 was already close to an optimum [18].

Next, we are interested in showing that the algorithm is robust to the choice of the temperature configuration. In Fig. 4 we plotted the average validation error obtained for each value of α considered in the inner cross-validation. The red lines denote the final error obtained by the ESN, with an interval of ± 0.0001 shown with dashed lines. We see that the error is practically constant in this interval, for values of α ranging from 0.1 to 0.6. Then, the error starts to increase for $\alpha > 0.6$. Probably, when the temperature is not decreasing fast enough, the algorithm is pruning also a large number of useful connections. However, there is a wide range of configurations of the scaling factor that results in the same testing error, providing a good robustness to the user's choice.

Regarding the number of connections, we show in Fig. 5 their final number in the reservoir. The numbers are averaged over the same validation folds and choices of the scaling

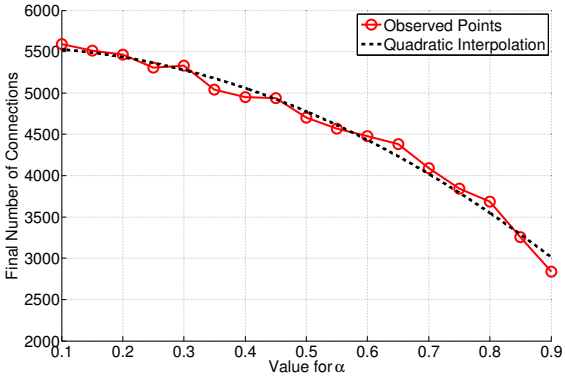

Fig. 5. Resulting number of connections for each choice of the scaling factor.

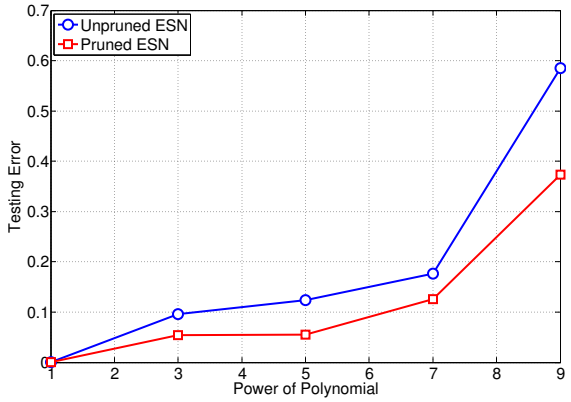

Fig. 6. MSE of the networks for increasing non-linearity of the task.

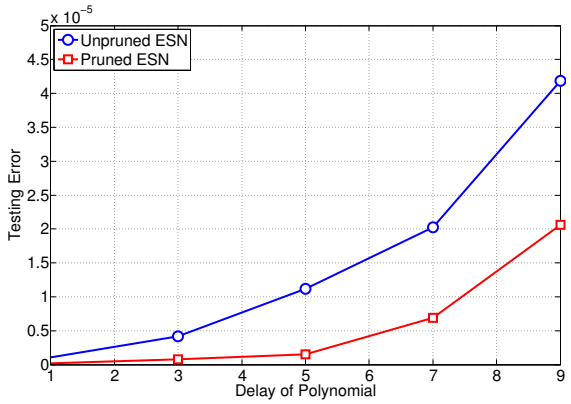

Fig. 7. MSE of the networks for increasing memory of the task.

factor considered in Fig. 4. Moreover, we also plotted a quadratic interpolation of the points, shown with a dashed black line. As we were expecting, the final number of connections is a monotonically decreasing function of the scaling factor.

B. Extended Polynomial

In the second set of experiments, our aim was to show that our pruning algorithm can additionally enhance the generalization capabilities of the network. Moreover, we wanted to investigate how the requirements of the task, in terms of both memory and non-linearity, affects the pruning operation, and *vice-versa*. To this end, we use the extended polynomial detailed in [7]. The input to the system consists in one random number extracted from a uniform distribution over $[-1, +1]$. The output is then defined as:

$$y(n) = \sum_{i=0}^{p} \sum_{j=0}^{p-i} c_{ij} u^i(n) u^j(n-d) \qquad (7)$$

where the coefficients c_{ij} are randomly distributed over the same distribution as the input data, and the two parameters p and d control the requirements of the task. In particular, increasing p has an effect on the order of the polynomial, hence augmenting the resulting non-linearity. Conversely, increasing d results in polynomial with higher delays, and thus higher memory requirements. We choose a reservoir similar to the one of Section V-A, but we increased the reservoir's size to $N = 250$. After some tests, we also increased the forgetting factor of the RLS to 0.998.

Following [7], we start by increasing the power of the polynomial from 1 to 9 by steps of 2, keeping fixed the delay at $d = 1$. The cooling factor α is not validated in this experiment, but is fixed at $\alpha = 0.95$, which was found to be an effective value for all the configurations. The MSEs of the pruned and unpruned ESNs are shown in Fig. 6. We see that the pruned version has a sustained gain in accuracy, and the gap remains constant for all configurations of the original polynomial. Thus, the pruning algorithm seems robust to an increase in the non-linearity implicit in the task.

Subsequently, we investigated the performance when increasing the delay of the polynomial from 1 to 9, keeping fixed the power at $p = 1$. By cross-validating, we saw that the best value for the scaling factor α was to be lowered to 0.3 for $d > 3$. In Fig. 7 we see the resulting MSEs of the networks, with a trend which closely follows that of the previous experiment, hence confirming the usefulness of the approach also for increasing levels of memory. It is interesting to observe that pruning had to be lowered for high levels of long-term memory of the reservoir, a finding which is very similar to the one detailed in [12].

Finally, the third set of experiments involved increasing simultaneously the power and the delay of the polynomial. In this case, α was set to 0.3 for $d = p > 3$ and subsequently lowered to 0.2 for $d = p = 9$. The results of the experiment, which are in line with the previous, are shown in Fig. 8.

The final number of connections for every reservoir considered thus far is shown in Fig. 9, where we show as a reference the original number of connections with a dashed black line. We see that the actual setting of p, d does not influence the final size, which is normal since our method does not depend explicitly on the desired output. The pruning strategy is extremely useful in removing synapses in all the situations, deleting more than 90% of their initial number whenever the scale factor is set to 0.95. Even in the worst

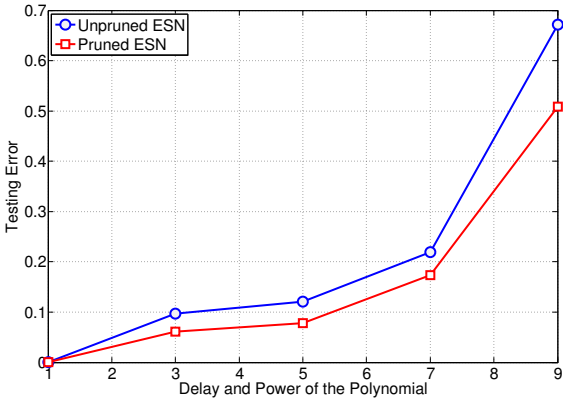

Fig. 8. MSE of the networks for increasing non-linearity and memory of the task.

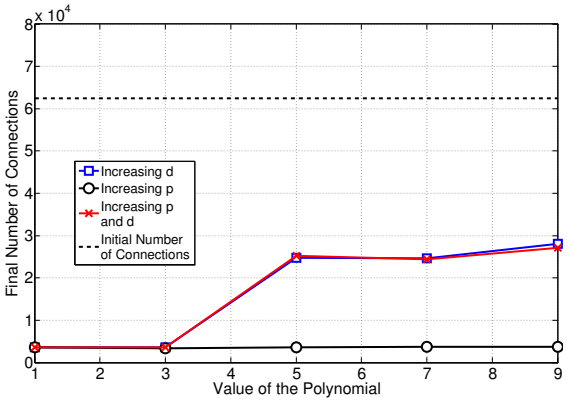

Fig. 9. Final number of connections of the reservoir.

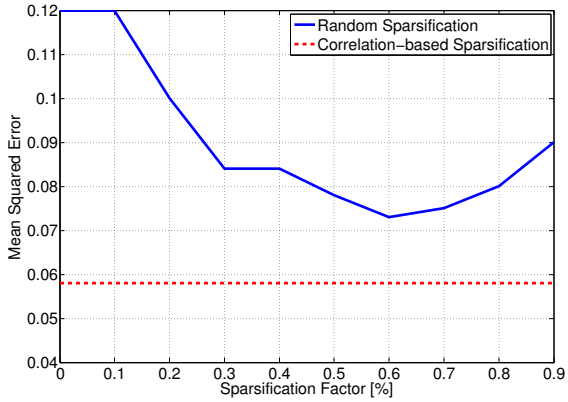

Fig. 10. Comparison between random pruning and our pruning strategy, in the setting most favorable to random pruning.

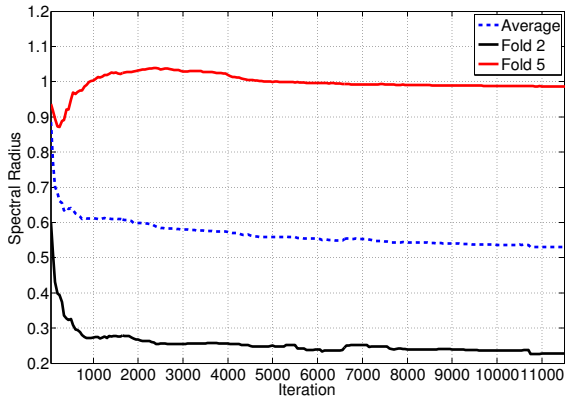

Fig. 11. Evolution of the spectral radius. We show the average over the different folds, together with the individual evolution over two representative folds.

case (α = 0.2 for $p, d = 9$), the final number of connections is $\approx$ 50% of the original one. Hence, our method seems highly robust in the case of different levels of non-linearity, while less pruning is required when increasing the long-term memory requested to the network.

An interesting final point is to compare our pruning strategy with random pruning of the reservoir during initialization. In Fig. 10 we show the error obtained by the random pruning strategy in the case that was found most favorable to it, i.e. $p = 5, d = 1$. We see that, in the best setting, with random pruning we are able to prune only 60% of the reservoir's connections, whilst there is a significant gap in performance with respect to our pruning strategy (denoted as "correlation-based" in Fig. 10). Moreover, as we analyzed in the previous section, the behavior of our strategy is more consistent when varying its parameters, while random pruning performance deteriorates rapidly when changing the sparsification factor.

C. Effect on the Spectral Radius

Before concluding the experimental section, we observe that our algorithm can result in a complex behavior with respect to the evolution of the spectral radius, which is not necessarily a decrease as is expected with a random deletion of the synapses. As an example, let us consider the experiment using the extended polynomial with $d = p = 1$. In Fig. 11 we plot the mean evolution of the spectral radius (the dashed blue line), averaged over the testing folds, along with the evolution in two particulars folds. The folds were chosen since they represented the most "extremes" behaviors.

We can see that, although the average evolution is toward a decrease of the original spectral radius, with an extreme of a final $\rho = 0.2$ in fold 2, in rarer cases we can witness an increase, as in fold 5 where the spectral radius assessed itself slightly lower than 1. Moreover, we can see that for a set of iterations the network had a spectral radius greater than unity [1]. This automatic tuning of the spectral radius is an interesting phenomenon which has been observed in several other supervised design strategies for reservoirs [6], that we plan to study in more detail in future works on the subject.

VI. Conclusion

In this paper we presented an algorithm for deleting connections in a randomly generated reservoir of an Echo State Network (ESN). The resulting ESN has always higher or comparable performance, but a smaller density of connections, resulting in lower computational and hardware requirements. We are currently working on combining our

pruning strategy with other existing approaches, such as those described in [6] and [12]. Moreover, we are interested in developing additional criteria, possibly leading to the direct pruning of neurons, or which are based on small neighborhoods of each synapse. Finally, we aim at proving some strict criteria to ensure the echo state property, and to fully automate the final pruning process.

ACKNOWLEDGMENTS

The authors would like to thanks Prof. John Butcher for sharing with us the original data of the extended polynomial used in Section V-B.

APPENDIX
RECURSIVE LEAST-SQUARE ALGORITHM

At time instant n, denote by $\mathbf{s}(n)$ the extended state $\mathbf{s}(n) = \left[\mathbf{u}(n)^T\ \mathbf{x}(n)^T\right]^T$, by $y(n)$ the desired scalar output (the extension to multi-valued output being straightforward), and by $\mathbf{W}(n-1)$ the current, available value of the output weights. The error is computed as:

$$e(n) = y(n) - \mathbf{W}(n-1)\mathbf{s}(n)$$

Denoting by λ the *forgetting factor* of the filter, a *gain vector* is computed as:

$$\mathbf{g}(n) = \mathbf{P}(n-1)\mathbf{s}(n)\left\{\lambda + \mathbf{s}^T(n)\mathbf{P}(n-1)\mathbf{s}(n)\right\}^{-1}$$

where $\mathbf{P}(n-1)$ is an approximation of the inverse of the cross-correlation of the input, and is updated as:

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{s}^T(n)\lambda^{-1}\mathbf{P}(n-1)$$

Finally, the weights are updated according to:

$$\mathbf{W}(n) = \mathbf{W}(n-1) + e(n)\mathbf{g}(n)$$

REFERENCES

[1] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, Aug. 2009.

[2] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," Technical Report GMD Report 148, German National Research Center for Information Technology, Tech. Rep., 2001.

[3] W. Maass, T. Natschl, and H. Markram, "A Model for Real-Time Computation in Generic Neural Microcircuits," in *Advances in Neural Information Processing Systems*, 2002.

[4] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, "Memory versus non-linearity in reservoirs," *2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, Jul. 2010.

[5] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, "Re-visiting the echo state property," *Neural Networks*, vol. 35, pp. 1–9, 2012.

[6] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, "Improving reservoirs using intrinsic plasticity," *Neurocomputing*, vol. 71, no. 7-9, pp. 1159–1171, Mar. 2008.

[7] J. Butcher, D. Verstraeten, B. Schrauwen, C. Day, and P. Haycock, "Reservoir computing and extreme learning machines for non-linear time-series data analysis." *Neural networks*, vol. 38, pp. 76–89, Feb. 2013.

[8] C. Omlin and C. Giles, "Pruning recurrent neural networks for improved generalization performance." *IEEE transactions on neural networks*, vol. 5, no. 5, pp. 848–51, Jan. 1994.

[9] M. Salmen and P. G. Ploger, "Echo state networks used for motor control," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2005, pp. 1953–1958.

[10] G. Chechik, I. Meilijson, and E. Ruppin, "Synaptic pruning in development: a computational account." *Neural computation*, vol. 10, no. 7, pp. 1759–77, Oct. 1998.

[11] G. Orlandi, F. Piazza, A. Uncini, and A. Ascone, "A biological approach to plasticity in artificial neural networks," in *1991 International Joint Conference on Neural Networks (IJCNN)*, vol. ii, 1991, pp. 583–586.

[12] X. Dutoit, B. Schrauwen, J. Van Campenhout, D. Stroobandt, H. Van Brussel, and M. Nuttin, "Pruning and regularization in reservoir computing," *Neurocomputing*, vol. 72, no. 7-9, pp. 1534–1546, Mar. 2009.

[13] D. Montgomery, E. A. Peck, and G. Vining, *Introduction to linear regression analysis*. Wiley, 2012, vol. 821.

[14] J. B. Butcher, C. R. Day, P. W. Haycock, D. Verstraeten, and B. Schrauwen, "Pruning reservoirs with random static projections," in *2010 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2010, pp. 250–255.

[15] J. Iglesias, J. Eriksson, F. Grize, M. Tomassini, and A. Villa, "Dynamics of pruning in simulated large-scale spiking neural networks." *Biosystems*, vol. 79, no. 1-3, pp. 11–20, 2005.

[16] G. A. Kochenberger *et al.*, *Handbook in Metaheuristics*. Springer, 2003.

[17] B. Schrauwen and J. Defour, "The introduction of time-scales in reservoir computing, applied to isolated digits recognition," in *International Conference on Artificial Neural Networks (ICANN 2007)*, 2007, pp. 471–479.

[18] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Advances in Neural Information Processing Systems*, 2003.

[19] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Neural networks with self-adaptive topology," *IEEE Transactions on Circuits and Systems*, pp. 1–6, 1991.

[20] Y. Nourani and B. Andresen, "A comparison of simulated annealing cooling strategies," *Journal of Physics A: Mathematical and General*, vol. 31, no. 41, p. 8373, 1998.

[21] A. Berman and X.-D. Zhang, "On the spectral radius of graphs with cut vertices," *Journal of Combinatorial Theory, Series B*, vol. 83, no. 2, pp. 233–240, 2001.