19th International Computer Science and Engineering Conference (ICSEC)
Chiang Mai, Thailand, 23-26 November, 2015

Fast and Effective Tuning of Echo State Network Reservoir Parameters Using Evolutionary Algorithms And Template Matrices

Sumeth Yuenyong
School of Information Technology, Shinawatra University
99 Moo 10 Bang Toey, Sam Khok District, Pathum Thani 12160, Thailand

***Abstract*—Echo State Network (ESN) is a special type of neural network with a randomly generated structure called the reservoir. The performance of ESN is sensitive to the reservoir parameters, which have to be tuned for best performance. Tuning of the reservoir parameters using evolutionary algorithms can be slow and produce inconsistent results. In this paper, we present a simple method for generating reservoirs based on templates that makes the reservoir matrices deterministic with respect to the parameters. Compared with the traditional method where the reservoir matrices are random, tuning of the reservoir parameters with an evolutionary algorithm needs less time, less number of cost function evaluations, and produces more reliable results using the proposed method.**

I. Introduction

Echo State Network (ESN) is a special type of recurrent neural network proposed by [1]. In order to obtain good results, the reservoir of the ESN structure must be generated with appropriate parameters [2], which has to be tuned by trial and error. For this reason, there have been interests in automating reservoir tuning by some sort of optimization. One common approach is to use evolutionary algorithms[1] to optimize the reservoir parameters. The problems with evolutionary algorithms is they are slow to run, and as will be seen, they are prone to noisy cost function caused by generating reservoirs randomly, producing inconsistent tuning results. In this paper, we present a simple method for generating reservoirs based on templates that make them effectively deterministic, leading to significant speedup of the average tuning times and more consistent tuning results.

A. Echo State Network

The basic idea of ESN is to use a large, recursive neural network excited by the input signal as a "reservoir". The state of the reservoir becomes the input to the "readout" layer, generally a linear combiner, that produces the final output. The entire structure is trained by adapting only the weights of the output layer, while the weights of the reservoir are randomly generated and held fixed, making training[2] of ESN a convex problem that can be solved by simple linear regression. This property had made ESN popular for modeling/identification of nonlinear dynamic systems [3], or prediction of signals generated by nonlinear processes/chaotic signals [4].

The ESN structure is shown in Figure 1. The reservoir is characterized by the input weight matrix $\mathbf{W}_{\text{in}}$ and the internal weight matrix $\mathbf{W}$. The output layer is characterized by the weight vector $\mathbf{w}_{\text{out}}$ (assuming single output for simplicity). Computation is performed in two steps: first the state of the reservoir is updated, and then the updated state vector is fed into the output layer to produce the final output. The state update equation is given by[3]

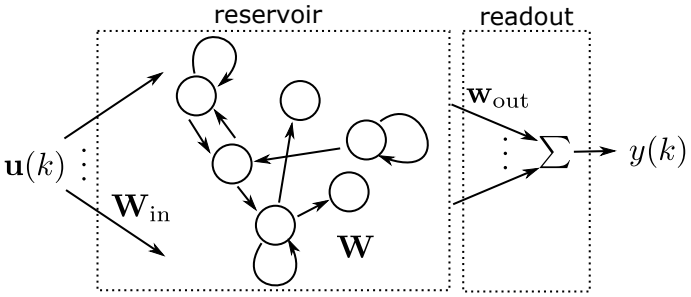

Fig. 1. Basic ESN structure. The circles represent neurons, generally with hyperbolic tangent activation function. The input weight matrix $\mathbf{W}_{\text{in}}$ is dense - every input is connected to every neuron. The internal matrix $\mathbf{W}$ may be dense or sparse. Both of these matrices are randomly generated. The output weight vector $\mathbf{w}_{\text{out}}$ (the readout) is dense - every neuron in the reservoir feeds the output layer. The output y is a simple weighted sum of the reservoir state (the output of each neuron at time k). During training, only $\mathbf{w}_{\text{out}}$ is adjusted to make y closer to the target signal.

$$\mathbf{x}(k) = \tanh\left[\mathbf{W}\mathbf{x}(k-1) + \mathbf{W}_{\text{in}}\mathbf{u}(k)\right] + \epsilon\mathbf{n}, \qquad (1)$$

where $\mathbf{x}(k-1)$ is the previous state vector, $\mathbf{x}(k)$ is the updated state vector, $\mathbf{u}(k)$ is the input vector and $\epsilon\mathbf{n}$ is the optional noise term. The output equation is given by

$$y(k) = \mathbf{w}_{\text{out}}^T\mathbf{x}(k). \qquad (2)$$

Batch training of the readout, given input $u(k)$ and target signal $t(k)$ both of length L is done by solving the following over-determined linear system in the least square sense

[1]These derivative-free global optimization algorithms are used for two reasons, the output of ESN is not differentiable with respect to the reservoirs parameters, and the error surface of these parameters is generally nonconvex

[2]Not to be confused with "tuning" which means adapting the parameters of the reservoir. "Training" means to adapt the weights of the output layer.

[3]For simplicity, we consider this simple state update in the original introduction of ESN [1] without the "leaky" parameter.

978-1-4673-7825-3/15/$31.00

$$\mathbf{P}\mathbf{w}_{\text{out}} = \mathbf{t}, \tag{3}$$

where $\mathbf{t}$ is a vector that holds all the L target signal samples and

$$\mathbf{P} = \begin{bmatrix} \mathbf{x}(1)^T \\ \mathbf{x}(2)^T \\ \vdots \\ \mathbf{x}(L)^T \end{bmatrix} \in L \times N, \tag{4}$$

where each row of $\mathbf{P}$ is the reservoir state vector at each time step and N denotes the reservoir size. The solution of (3) is usually obtained by ridge regression.

B. Generating The Reservoir

The parameters involved in generating the individual weights of $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ are:

1) The distribution from which to draw the elements of $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$.
2) The spectral radius[4] of $\mathbf{W}$, denoted as $\tilde{\rho}$.
3) The scaling of the input weight matrix $\mathbf{W}_{\text{in}}$, denoted as s.
4) The reservoir size N.
5) The sparseness of $\mathbf{W}_{\text{in}}$.

Once the five parameters have been selected, reservoir generation proceed as follows:

1) Sample the nonzero elements of $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ from the chosen distribution.
2) Scale the input matrix: $\mathbf{W}_{\text{in}} = s\mathbf{W}_{\text{in}}$
3) Let $\rho(\mathbf{W})$ denote the spectral radius of the matrix $\mathbf{W}$. Scale $\mathbf{W}$ to have a specific spectral radius by: $\mathbf{W} = \tilde{\rho}\mathbf{W}/\rho(\mathbf{W})$

The scaled $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ are respectively the input and the internal reservoir matrices that will be used in the ESN model. Note that compared to regular recurrent neural networks, the reservoir size N is quite large, with values that can often be in the hundreds or even thousands.

II. Tuning Reservoir Parameters with Evolutionary Algorithms

A. The Choice of The Parameters to Optimize

Out of the above five reservoir parameters, some study have proposed to optimize all of them [5], some have even proposed optimizing the joint parameter space between the reservoir and the readout weights [6], [7]. This is not a good approach, as evolutionary algorithms become less effective as the dimension of the parameter space increases. The chance of finding the optimal region becomes smaller, and larger population size and longer running time are needed. It is better for both efficiency and computation to concentrate only on the parameters that have strong impact on the performance of ESN.

The distribution to be sampled from has virtually no impact and can be chosen as the convenient $[-1, 1]$. Sparse $\mathbf{W}$ usually performs just slightly better than dense $\mathbf{W}$. The main advantage of using sparse $\mathbf{W}$ is less computation, so highly sparse $\mathbf{W}$ is desired. However, if the sparseness is above 90%, eigenvalue calculation for $\mathbf{W}$ may sometime fail to converge. So as a rule of thumb, if sparse $\mathbf{W}$ is used, the sparseness should be set to no more than 90%. The reservoir size N is not really a parameter, but a design choice. It is trade-off between better performance from a larger reservoir, against available training data and computational resource. The critical parameters to optimize are the input scaling s and the spectral radius $\tilde{\rho}$, because they have strong impact on the performance [8].

[4]The spectral radius of a matrix is the maximum of the magnitudes of its eigenvalues.

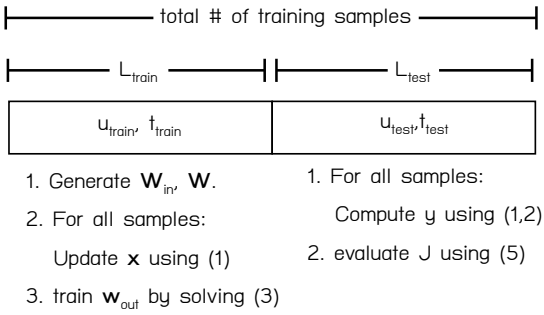

Fig. 2. An illustration of how the cost function for reservoir tuning is evaluated. The available samples are split into two blocks. The first block is used to train the readout weights, and the second block is used to evaluated the performance of the trained ESN structure.

B. The Cost Function for Reservoir Tuning

In this section we will drop the time index k to avoid cluster. Regardless of what evolutionary algorithm is used, the mechanism of evaluating the cost function is the same. This is illustrated in Figure 2. At each point in the reservoir parameter space that the algorithm is searching, the parameters at that point is used to generate the reservoir matrices $\mathbf{W}_{\text{in}}, \mathbf{W}$. The reservoir is then fed with u_{train} and the state vector $\mathbf{x}$ is calculated at each time step to construct the matrix $\mathbf{P}$, concurrently, each sample of t_{train} is stored in the vector $\mathbf{t}$. Once all the L_{train} samples had been used, the readout is trained by solving (3). The performance of the trained ESN structure is then evaluated by feeding it with u_{test}, calculating y_{test}, and then comparing y_{test} versus t_{test} by some sort of error measure. Thus we can write the cost function as

$$J(\mathbf{v}) = f(y_{\text{test}}, t_{\text{test}}), \tag{5}$$

where $\mathbf{v}$ denote the parameter vector at a point and y_{test} is obtained as above. The function f can be the mean square error (MSE) or its normalized version. The reservoir tuning problem can then be posed as follows

$$\underset{\mathbf{v}}{\operatorname{argmin}} \left[J(\mathbf{v}) \right]. \tag{6}$$

Practically, it is better to use as f the normalized mean square error (NMSE) or the normalized root mean square error (NRMSE). The reason being because the value of J will fall in the same range across different input and target signals, allowing one to set a fixed tolerance value as the termination criterion for the evolutionary algorithm.

Upon casual inspection, the cost function (5) may seem simple to evaluate, just train and evaluate ESN at each point $\mathbf{x}$

that the algorithm calls for. However, there is a pitfall. Due to the fact that generating the reservoir involves sampling from a random distribution, even with exactly the same set of parameters, different reservoirs will be generated each time. These different reservoirs can have significantly different performance [9]. This means that $J(\mathbf{v})$ will be different each time it is evaluated even if $\mathbf{v}$ remains the same, i.e., (5) evaluated in this manner is noisy. This causes convergence to take longer (larger number of cost function evaluations needed), or sometimes mis-convergence altogether.

Moreover, it can be seen from the steps of generating reservoirs that eigenvalue calculation of a $N \times N$ matrix is needed to determine $\rho(\mathbf{W})$. This needs to be computed every time J is called, resulting in a lot of overhead. In the next section, we present a simple method that eliminates both noisy J and overhead from repeated eigenvalue calculations.

III. Generating Reservoirs Using Templates

We propose the use of "templates" for generating reservoirs. The idea is to randomly generate reservoir matrices only once, store these matrices in memory and modify them in a deterministic way based on s and $\tilde{\rho}$. That is, *before* starting reservoir tuning, generate "template matrices" $\widehat{\mathbf{W}}$ and $\widehat{\mathbf{W}_{\text{in}}}$ by sampling from the chosen distribution, calculate the spectral radius $\rho(\widehat{\mathbf{W}})$ and keep the result in memory. Then these templates can be transformed to actual reservoir matrices with input scaling s and spectral radius $\tilde{\rho}$ simply by

$$\mathbf{W} = \frac{\widehat{\mathbf{W}}}{\rho(\widehat{\mathbf{W}})}\tilde{\rho} \quad \mathbf{W}, \widehat{\mathbf{W}} \in N \times N \tag{7}$$

and

$$\mathbf{W}_{\text{in}} = s\widehat{\mathbf{W}_{\text{in}}} \quad \mathbf{W}_{\text{in}}, \widehat{\mathbf{W}_{\text{in}}} \in N \times l, \tag{8}$$

where $\widehat{\mathbf{W}}$ and $\widehat{\mathbf{W}_{\text{in}}}$ are respectively the template for the input matrix and the internal matrix and l is the dimension of the input signal. It is clear that this approach only works if we limit the reservoir parameters to only s and $\tilde{\rho}$. There is no way to make a template if any of the remaining 3 parameters can change.

There are two main advantages to this approach. First, since the templates are generated only once before we start the evolutionary algorithm, for any $s, \tilde{\rho}$, *exactly* the same $\mathbf{W}, \mathbf{W}_{\text{in}}$ will be generated. This completely eliminates the noise in the value of J, since the reservoir matrices are no longer random, but deterministic transformations of two matrices that already exist. The second advantage is that eigenvalue calculation has to be performed only once on $\widehat{\mathbf{W}}$, instead of every time J is called, on $\mathbf{W}$. This saves a lot of computation since eigenvalue decomposition of a large matrix is an expensive operation. Algorithm 1 and 2 compare traditional reservoir generation vs. our proposed template method. It can be seen that in Algorithm 2, the return matrices are deterministic with respect to the parameters $s, \tilde{\rho}$ being optimized. No random sampling and no eigenvalue decomposition are performed inside the procedure.

Algorithm 1 Generating Reservoirs without Templates

```
procedure GenReservoir(s, ρ̃)
    Generate W, W_in
    Calculate ρ(W)
    W_in ← s W_in
    W ← (W / ρ(W)) ρ̃
    return W, W_in
end procedure
```

Algorithm 2 Generating Reservoirs Using Templates

```
procedure GenReservoir(s, ρ̃, Ŵ_in, Ŵ, ρ(Ŵ))
    W_in ← s Ŵ_in
    W ← (Ŵ / ρ(Ŵ)) ρ̃
    return W, W_in
end procedure
```

IV. Experimental Results

We simulated two problems in order to confirm the effectiveness of the proposed method. The first one is the identification of a 30th order nonlinear recurrent system, called the "narma30" dataset in [10]. The second one is the prediction of a laser time-series from the Santa-Fe time-series competition [11]. The two problems are referred to as "narma30" and "laser" respectively. We choose the Covariance Matrix Adaptation Evolution Strategy (CMAES) algorithm [12] over other evolutionary algorithm because it is effective with small population size λ (default value of $4 + 3\log_{10}(D)$ vs. $3D$ or $10D$ for most evolutionary algorithms). Moreover, it has only a single parameter to set, the initial standard deviation of the (initially diagonal) Covariance matrix σ. The other "internal" parameters are determined by the algorithm itself. The CMAES algorithm works by creating a multi-variate Gaussian distribution with mean located at the current best point found, drawing test points to evaluate from this pdf and choosing the best one to be the mean of the next iteration's pdf, while concurrently adapting the Covariance matrix of the distribution such that its shape matches the shape of the error surface around the current best point. For an in-dept introduction to CMAES, please refer to the tutorial article in [13]

We set the reservoir size to be $N = 100$ with dense $\mathbf{W}$ and $L_{\text{train}} = L_{\text{test}} = 500$. For the error measure f in (5), we used the NRMSE. For each problem we simulated 50 independent tuning trials of both traditional reservoir generation and our template-based method. The parameter and options of the CMAES were set as follows: the starting point was $s = 0.5$ and $\tilde{\rho} = 0.8$, the initial variance was 0.66, the tolerance in the cost function value for terminating the algorithm was 0.05, the population size was 5 and maximum number of iterations to run was 200. The bounds on the search space was $[0.01, 1]$ for s and $[0.1, 1]$ for $\tilde{\rho}$.

Figure 3 compares the results for the "narma30" problem, with and without using templates. The error surface contour plots were generated by calculating J at 10,000 points in the parameter space. The crosses indicate the optimal paramters found by the CMAES algorithm for each trial. Figure 4 shows the zoomed in of the region around the bottom right hand corner of Figure 3. The zoomed-in figure reveals that the global minimum region is consistently found only when templates are used.

Figure 5 shows the results for the "laser" problem. In this case it can be seen that the error surface is more challenging than the previous problem, it has more variation and the region with the minimum NRMSE is disjoint. Using templates, the region with the minimum NRMSE was still consistently found, only two crosses lie outside the region. Without using templates however, the best parameters found are scattered all over the parameter space. This shows that the noise in the cost function values described in Section 2B affects the ability of evolutionary algorithms to effectively search the parameter space.

From the results shown by Figures 4 and 5, it can be seen that the proposed template-based reservoir generation leads to much better tuning result. The final location in the parameter space found by the CMAES algorithm were consistently located inside the global minimum region of both test problems.

A. Statistical Analysis of Average Tuning Time And Number of Function Evaluations

Besides from better and more consistent tuning results, the use of templates also resulted in significant speedup in the average tuning time. We statistically compared the average times[5] in seconds it took to complete tuning, and the average number of cost function evaluation (feval) by using independent two-samples t-test with the following hypothesis

$$\begin{aligned} H_0 &: \mu_{t_{\text{template}}} = \mu_{t_{\text{traditional}}} \\ H_A &: \mu_{t_{\text{template}}} < \mu_{t_{\text{traditional}}} \end{aligned}, \tag{9}$$

and

$$\begin{aligned} H_0 &: \mu_{\text{feval}_{\text{template}}} = \mu_{\text{feval}_{\text{traditional}}} \\ H_A &: \mu_{\text{feval}_{\text{template}}} < \mu_{\text{feval}_{\text{traditional}}} \end{aligned}, \tag{10}$$

The statistical analysis result is shown in Table I. In terms of average tuning time, the use of templates resulted in statistically significant speedup for both problems, especially for the laser problem where the speedup was one order of magnitude. In terms of the number of cost function evaluations, there was no statistically significant difference between our method and the traditional method for the narma30 problem. For the laser problem, the average number of function evaluations was reduced by about 5 times. This shows that our proposed method can achieve large reduction in both tuning time and number of function evaluations. The reason for the large reduction of the number function evaluations needed for the laser problem when templates are applied is likely due to the fact that the error surface of the laser problem has more variation compared to that of the narma30 problem, which is mostly flat. On a flat error surface, the noise from evaluating the cost function without using templates tends to cancel out from point to point since it is a zero mean random variable. However, on an error surface with more variations, the noise do not cancel, so the cost function remain noisy. It is normal for evolutionary algorithms to need more function evaluations on noisy surfaces than a noise-free ones.

The computational time for generating the templates can be broken down as follows: the time it takes to sample the elements of $\mathbf{W}$ and $\mathbf{W}_{\text{in}}$ and the time it takes to perform eigenvalue decomposition on $\mathbf{W}$. The first one is essentially constant time, while the second one takes less than one second even with reservoir sizes of thousands of neurons, thus the time it takes the generate the templates is less than one second in our set of tests. Compared to the reduction in tuning time achieved by the use of templates which can be an order of magnitude as shown by Table I, the time it takes to generate the templates themselves is very short and well worth the effort.

[5]Of course, running in the same environment and on the same machine.

TABLE I. STATISTICAL ANALYSIS AVERAGE TUNING TIME AND NUMBER OF FUNCTION EVALUATIONS.

	Average tuning time (s.)		
	template	traditional	p-value
narma30	2.63	7.36	$9.51E^{-9}$
laser	5.67	36.94	$< 2.2E^{-16}$

	Average number of function evaluations		
	template	traditional	p-value
narma30	127.32	113.46	0.8726
laser	118.62	539.52	$1.83E^{-15}$

V. CONCLUSION

In this paper we have presented a method for generating reservoirs based on templates that eliminates the noise in the cost function for reservoir tuning, as well as the need to repeatedly solve large eigenvalue problems. This was achieved by pre-generating templates for the reservoir matrices, and scaling them each time the cost function is evaluated, instead of generating the reservoir matrices from scratch. In this manner, reservoir matrices become deterministic for a given set of templates, and thereby eliminates noise in the cost function, as well as reducing the computational overhead of each function call.

Experiments show that our approach leads to better and more consistent tuning results, as well as 3-7 folds speedup in the average tuning time, especially for a problem that has a complicated error surface where the average number of function evaluation needed was reduced by almost five times. Further work includes more comprehensive evaluations by conducting simulations on more test problems, as well as for different reservoir sizes and training signals lengths.

REFERENCES

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

[2] ——, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik, 2002.

[3] ——, "Adaptive nonlinear system identification with echo state networks," *Networks*, vol. 8, p. 9, 2003.

[4] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[5] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, "Training recurrent networks by evolino," *Neural computation*, vol. 19, no. 3, pp. 757–779, 2007.

[6] F. Jiang, H. Berry, and M. Schoenauer, "Supervised and evolutionary learning of echo state networks," in *Parallel Problem Solving from Nature–PPSN X*. Springer, 2008, pp. 215–224.

[7] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

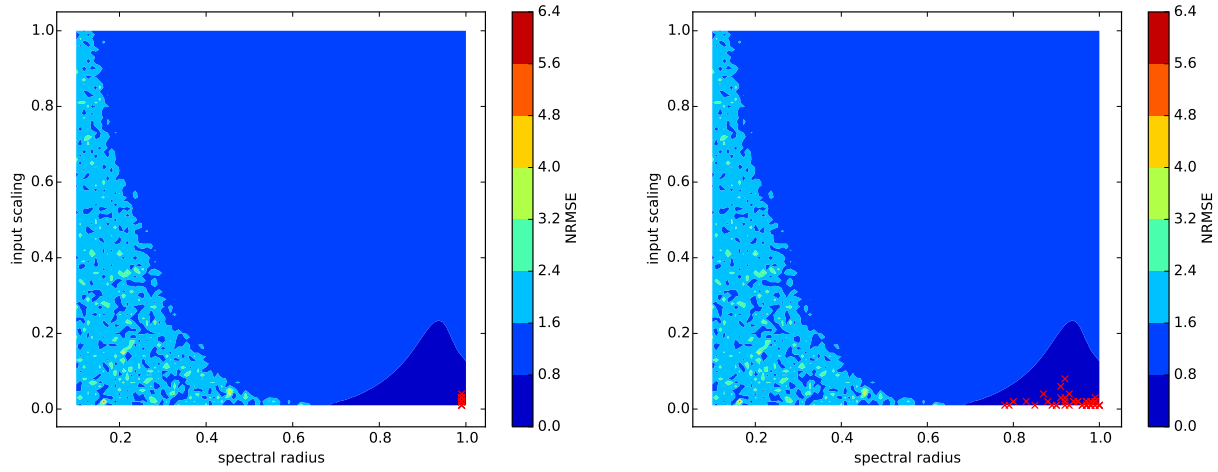

Fig. 3. The best parameters found (the crosses) by the CMAES algorithm over 50 independent trials for the "narma30" problem, (left) using our proposed template method and (right) without using template. It seems that in both cases the region with the minimum NRMSE was consistently found, but see Figure 4 for a zoom-in around the bottom right corner.

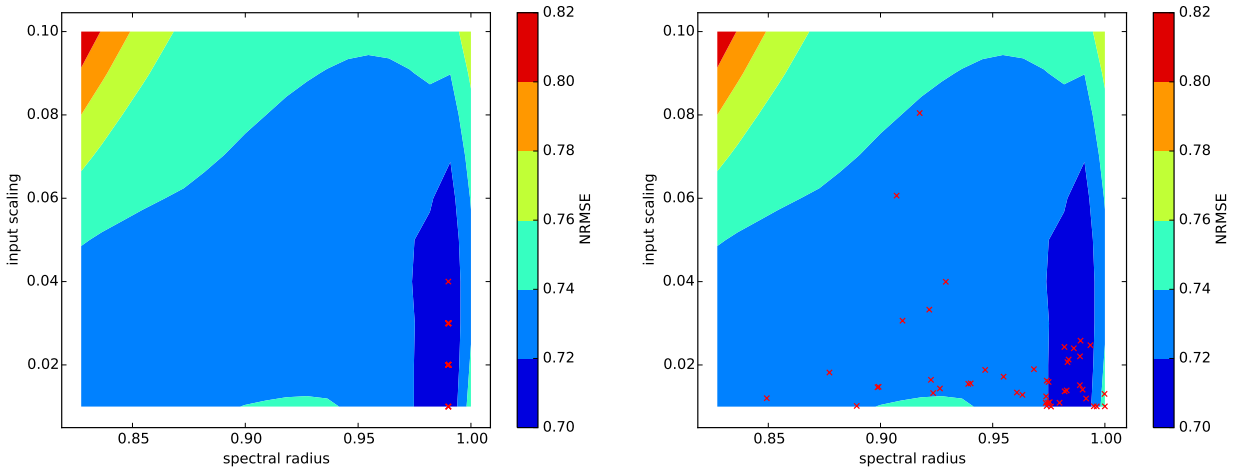

Fig. 4. The bottom right corner from Figure 3 zoomed in. The finer scale on the NRMSE now reveals that the global minimum region is consistently found only when templates are used (left). When templates are not used (right) the optimal location found by CMAES are scattered in a wider area and are not consistently located in the global minimum region.

[8] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 659–686.

[9] M. C. Ozturk, D. Xu, and J. C. Príncipe, "Analysis and design of echo state networks," *Neural Computation*, vol. 19, no. 1, pp. 111–138, 2007.

[10] D. Verstraeten, B. Schrauwen, S. Dieleman, P. Brakel, P. Buteneers, and D. Pecevski, "Oger: modular learning architectures for large-scale sequential processing," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2995–2998, 2012.

[11] N. Gershenfeld and A. Weigend, "Santa fe time series competition data," Standford University, 1994. [Online]. Available: http://www-psych.stanford.edu/ãndreas/Time-Series/SantaFe.html

[12] N. Hansen, "Cma-es - a stochastic second-order method for function-value free numerical optimization," October 2011. [Online]. Available: https://www.lri.fr/ hansen/msrc-cmaes-nov-2011.pdf

[13] ——, "The cma evolution strategy: A tutorial," *Vu le*, vol. 29, 2005.

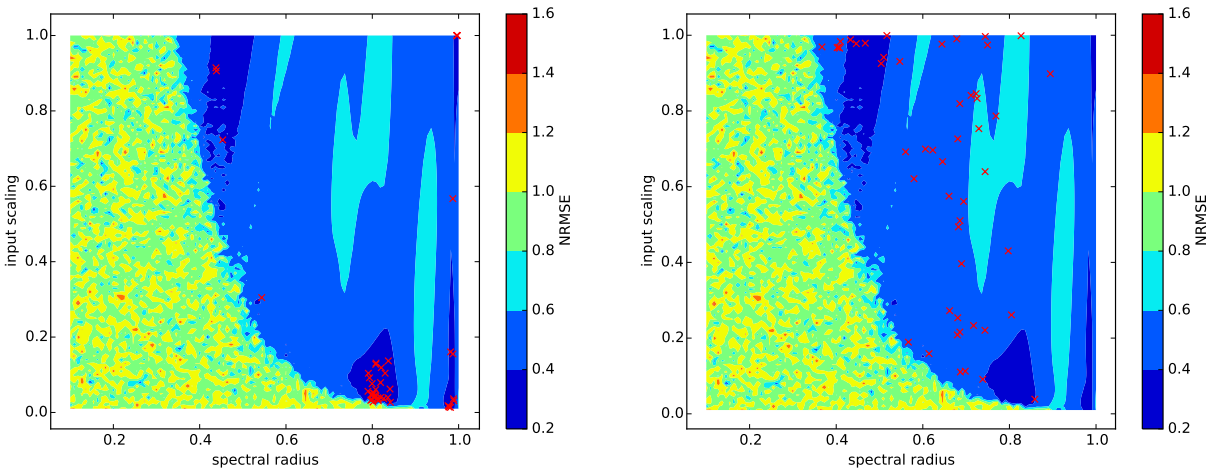

Fig. 5. The best parameters found (the crosses) by the CMAES algorithm over 50 independent trials for the "laser" problem, (left) using our proposed template method and (right) without using template. The error surface is more challenging than the previous problem, it has more variation and the region with the minimum NRMSE is disjoint. Using templates, the region was still consistently found. Without templates however, the best parameters found are scattered all over the parameter space.