Automatic Construction of Radial-Basis Function Networks Through an Adaptive Partition Algorithm

Ricardo Ocampo-Vega[1], Gildardo Sanchez-Ante[1(✉)], Luis E. Falcon-Morales[1], and Humberto Sossa[2]

[1] Tecnologico de Monterrey, Campus Guadalajara, Av. Gral Ramon Corona 2514, 45201 Zapopan, Jal, Mexico
gildardo@itesm.mx

[2] Instituto Politecnico Nacional-CIC, Av. Juan de Dios Batiz S/N, Gustavo A. Madero, 07738 Distrito Federal, Mexico

Abstract. Radial-Basis Function Neural Networks (RBFN) are a well known formulation to solve classification problems. In this approach, a feedforward neural network is built, with one input layer, one hidden layer and one output layer. The processing is performed in the hidden and output layers. To adjust the network for any given problem, certain parameters have to be set. The parameters are: the centers of the radial functions associated to the hidden layer and the weights of the connections to the output layer. Most of the methods either require a lot of experimentation or may demand a lot of computational time. In this paper we present a novel method based on a partition algorithm to automatically compute the amount and location of the centers of the radial-basis functions. Our results, obtained by running it in seven public databases, are comparable and even better than some other approaches.

Keywords: Radial-basis functions · Neural networks · Adaptive parameter adjustment · Classification

1 Introduction

We humans have the ability to quickly identify the face of a friend among many people, to distinguish oranges from apples in the market and to remember the name of a song playing in the radio. All those tasks could be considered as instances of what is called the *classification* problem. Solving it implies assigning a certain label -the class- to a set of features. There are quite a few methods to solve classification tasks. Those methods can be broadly grouped in supervised or unsupervised. The *supervised* approach tries to find an approximate model for the classes, whose error is smaller than a certain threshold. In order to do that, it is required to have a set of solved instances of the problem. Those instances are used to improve an initial model through a *training algorithm* [1].

J.F. Martínez-Trinidad et al. (Eds.): MCPR 2016, LNCS 9703, pp. 198–207, 2016.
DOI: 10.1007/978-3-319-39393-3_20

One of the most well known supervised methods is the Artificial Neural Networks (ANN). Current ANN models are used to solve complex problems, such as image processing in medical applications [2], the control of devices [3], as well as face [4] and speech recognition [5], to mention just a few [6]. There are different types of ANNs. One of them is called the Radial-Basis Function (RBF) Neural Networks.

Radial-Basis Function Neural Networks (RBFNN) require three layers of artificial neurons: the first layer connects the network with the environment, the second layer is the only hidden layer, usually implemented by radial activated functions. The third layer is the output, which is implemented as a weighted sum of the outputs of the hidden units (that is, a linear function) [7]. Figure 1 illustrates a RBF NN with three inputs, four hidden units and one output. RBFs are important given their capability to model non-linear systems with only two layers. Under certain conditions on the shape of the activation functions, RBFNN are universal approximators [8]. That means that such networks can approximate any continuous function with arbitrary precision, if enough hidden neurons are provided. Other ANN models, like the Multi-Layer Perceptron, can also approximate non-linear functions, but they may require multiple intermediary layers [7].

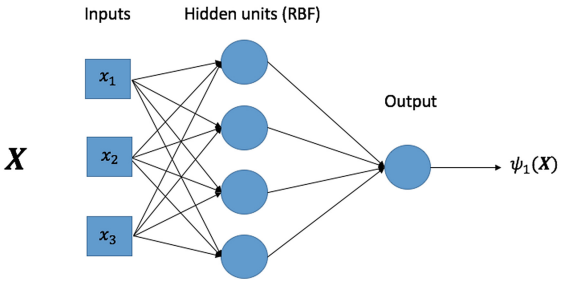

Fig. 1. Architecture of a radial basis function network.

When RBF networks are used in classification problems, the inputs are associated to the features that characterize the problem, and the outputs represent the classes or labels. The hidden units correspond to subclasses. Several different functions have been considered as activation functions for the hidden neurons [9]. Undoubtedly, the Gaussians are the most common ones. Equation 1 presents the definition of a Gaussian function.

$$\boldsymbol{\phi}_j(\mathbf{X}) = \exp[-(\mathbf{X} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{X} - \boldsymbol{\mu}_j)] \tag{1}$$

for $j = 1, \ldots, L$, where $\mathbf{X}$ is the feature vector, L is the number of hidden units, μ_j and Σ_j are the mean and the covariance matrix of the j-th Gaussian function.

As for the output layer, it is computed through a weighted sum, defined in Eq. 2:

$$\boldsymbol{\Psi}_k(\mathbf{X}) = \sum_{j=1}^{L} \lambda_{jk}\phi_j(\mathbf{X}) \tag{2}$$

for $k = 1, \ldots, M$, where λ_{jk} are the output weights, M is the number of output units, and ϕ_j represents node j in the hidden layer. It is common that for pattern classification problems the output is limited to the interval $(0, 1)$ by a sigmoidal function.

The training of the RBF neural networks is often performed in two steps: the first step is to find the number of neurons in the hidden layer, which means defining the corresponding radial-basis functions. The second step consists in finding the weights for the connections to the output layer. Despite the importance of RBF Networks, it turns out that creating one to solve a particular problem is an issue that may imply a good amount of experimental work. One traditional method to perform those steps consists in first deciding on the number L of choosing hidden neurons. Then choosing arbitrarily L data points as centers. Such procedure often causes ill-conditioning, but it has been shown that for large training sets, this approach gives reasonable results [1]. Another way to determine the number of neurons in the hidden layer is performed by using clustering (K-means). Still, it is up to the person solving the problem to decide the number L of neurons. Once that has been done, the K-means algorithm finds the centers of each cluster. This is an iterative process that implies solving an optimization problem. From that, is possible to define the radial-basis functions $\boldsymbol{\phi}_j(\mathbf{X}); \forall j \in \{1, 2, \ldots, L\}$. Then, the weights for the output layer are computed, usually through Least-Means Squares (LMS) or Recursive-Means Squares (RMS). Since the number of neurons (centers) is defined arbitrarily, usually several values are tested to determine with which the RBFN gives the best results.

In this work, we propose a new method to automatically determine the number of neurons on the hidden layer. Moreover, our method adjusts the number of neurons so that we get the best performance in the RBFN for a given training dataset. Our contribution is to provide a simple algorithm to construct the best RBFNN for a given problem, requiring no intervention from the user. The experimental results show that our approach is competitive in practice compared with RBFNN generated with the traditional trial and error process. The remainder of the paper is organized as follows: Sect. 2 describes other works related with ours. Section 3 presents our method. Section 4 describe the experiments and results and Sect. 5 discusses possible future work.

2 Related Works

There is a lot of research in neural networks and their applications. In particular, some researchers have attempted to develop methods to automatically choose some of the parameters that control the functioning of the RBFNs. For instance,

Moody et al. [10] analyzed three versions of a learning algorithm, using nearest neighbor prediction, adaptive processing units with LMS and self-organizing adaptive units. According to their results, their proposal allows to train faster than with backpropagation.

In another approach, Poggio et al. [11] introduce a supervised method based on regularization to find the centers of the radial-basis functions and to update the weights. In a way, their work leads to a generalization of RBFNs. In this work, the authors focus on the mathematical formulation of their approach and no information is given regarding the applicability of it to practical problems.

Chen et al. [9] on the other hand, consider the application of Orthogonal Least-Squares (OLS). Under this approach, the functions are chosen one at a time until an appropriate network is built. The center of each Gaussian is given by one of the data points. The points are chosen by constructing a set of orthogonal vectors. The authors show several applications and mention as further work a comparison with Moody's algorithm [10].

Kurban et al. [12] introduces a comparison of training algorithms of radial basis function (RBF) neural networks for classification purposes. The authors compare: gradient descent (GD), genetic algorithms (GA), Kalman filtering (KF) and Artificial Bee Colonies (ABC). They use datasets from UCI to test the algorithms. In their results, ABC offered a good performance, so they used that implementation to solve a problem related with the classification of terrains given information from inertial sensors. The main issue with the approach is that in general the convergence time is bigger than for methods such as GD and KF.

Chun-Tao et al. [13] introduce a methodology based on Particle Swarm Optimization (PSO) and simulation (Resource allocation), to train RBF Networks. Their results seems to suggest to use the method with caution since it got trapped in local minima, though. Oh et al. [14] extended this work and use fuzzy logic to help in the process. Their results are comparable to previous works.

In a very recent work, Reiner [15] propose two algorithms to construct RBF networks. One of them is based on an Incremental Extreme Machine Learning added with a simplex optimization. The second one uses a Levenberg-Marquardt algorithm to find the locations and configurations of the Radial-Basis functions. What the author finds is that doing this allows to define more compact networks with smaller errors, compared with ten different algorithms.

3 Methodology

The classic training method for RBFN is hybrid, and it is comprised by two steps. In the first step, the parameters of the hidden nodes are set. Afterwards, the weights of the output layer are adjusted until a halt criteria is satisfied. For this method, the k-means algorithm is used to set the parameters of the Gaussian nodes. The k-means algorithm find L clusters with minimum internal variation, but with maximum variation among clusters. The number of clusters (L) must be manually configured, and the method does not guarantee halting in a global minimum solution. When k-means finishes, a network is created considering each cluster as a hidden layer node, and a single output layer node.

The proposed method is also hybrid, and it is performed in two steps as well. However, we use the SMLP-P algorithm [16] to set the number of clusters L and the radial basis functions parameters. We adjust the weights of the output layer employing the Recursive Least Squares (RLS) algorithm. We call our method RBFN-SLMP.

3.1 Estimation of Hidden Layer Parameters

In the hidden layer, it is required to set three kinds of parameters: (1) number of Gaussian nodes L, (2) center of each Gaussian function $\mathbf{x}_j$ and (3) amplitude σ of the Gaussian functions $\varphi_j(\mathbf{x}_j, \sigma)$. It is proposed to use SLMP-P [16] to estimate the latter. Originally, this algorithm is used to train the Lattice Neural Networks with Dendrite Processing (LNNDP) [17]. With this approach, we can automatically calculate the L Gaussian nodes, and their parameters $\mathbf{x}_j$ and σ. Although, the amplitude of the Gaussian nodes is not uniform for all of them, we calculate a general parameter as Haykin suggests in [7]. The proposed algorithm of training guarantees to stop in a global minimum.

Given p classes, C^k, $k = 1, 2, 3..., p$, each of them with n features, the training algorithm is the following:

1. A hypercube HC^n is created. It contains all the cases of the training set. Looseness is added to each of the HC sides in order to add tolerance to noise.
2. HC^n is divided into 2^n equal sized, smaller hypercubes.
3. For each HC, it is verified if there exist at least two observations with different classes. If it is the case, it returns to step 2; otherwise, it goes to step 4.
4. For each HC, $\mathbf{x}_j$ is calculated. Also, σ is calculated.
5. A RBFN is created with L hidden nodes, where L is the number of HC obtained.

The functions for σ and center($\mathbf{x}_j$), based on [7] are:

$$\sigma = \frac{d_{max}}{\sqrt{2L}} \tag{3}$$

$$\text{center } (\mathbf{x}_j) = \frac{1}{|S_j|} \sum_i \mathbf{x}_{j_i} \tag{4}$$

where d_{max} is the maximum Euclidean distance among the center of the clusters and $|S_j|$ is the cardinality of cluster S_j that contains all observations $\mathbf{x}_{j_i}$.

3.2 Estimation of Output Layer Weights

The hidden layer's training is recursive, therefore, Haykin [7] suggests that the computation of the weights in the output layer should be recursive too. Thus, we use *Recursive Least Squares* (RLS) to adjust the weights. The training algorithm is the following:

1. Begin with $\hat{\mathbf{w}}(0) = \mathbf{0}$ and $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. In this case, we use an initialization to zero for the weights, as in [7,18].
2. Given a sample $\{\boldsymbol{\phi}(i), d(i)\}_{i=1}^{N}$ calculate the following for $n = 1, 2, ..., N$:
   $\mathbf{P}(n) = \mathbf{P}(n-1) - \frac{\mathbf{P}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{P}(n-1)}{1+\boldsymbol{\phi}^T(n)\mathbf{P}(n-1)\boldsymbol{\phi}(n)}$
   $\mathbf{g}(n) = \mathbf{P}(n)\boldsymbol{\phi}(n)$
   $\alpha(n) = d(n) - \hat{\mathbf{w}}^T(n-1)\boldsymbol{\phi}(n)$
   $\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}^T(n-1) + \mathbf{g}(n) * \alpha(n)$
3. When $MSE < \theta$ halt.

where θ is manually defined by the user, λ is a positive constant as small as possible, and $d(i)$ is the class of the sample i. The vector $\boldsymbol{\phi}(i)$, results after the transformation in the hidden layer. It is defined as,

$$\boldsymbol{\phi}(i) = [\varphi(\mathbf{x}_i, \boldsymbol{\mu}_1), \varphi(\mathbf{x}_i, \boldsymbol{\mu}_2), ..., \varphi(\mathbf{x}_i, \boldsymbol{\mu}_L)]^T \tag{5}$$

$\boldsymbol{\mu}_j$ is the hidden node center j. The Gaussian function is expressed as follows:

$$\varphi(\mathbf{x}_i, \boldsymbol{\mu}_j) = exp\left(-\frac{1}{2\sigma_j^2}||\mathbf{x}_i - \boldsymbol{\mu}_j||^2 \right),\ j = 1, 2, ..., L \tag{6}$$

4 Experiments and Results

The experiments were run in a server with processors Xeon E5-2650 (20M Cache, 2.00 GHz, 8.00 GT/s Intel QPI) with a shared memory of 100GB. The OS was CentOS 6.5. The code is programmed in Python 3.3.3 with embedded code in R 3.0.3. We tried with 25, 50, 75, 100 and 300 iterations for RLS algorithm. The λ was selected as small as possible.

4.1 Databases

We used the benchmarks published on the Center for Machine Learning and Intelligent Systems (UCI) [19]. The benchmarks were selected following the recommendation of Babu and Suresh in [20]. Table 1 shows for each dataset the amount of features it has, the number of classes and the number of samples used for training and testing. In total, we apply our method to seven different datasets comprising up to a few dozens of features and a few hundreds of samples.

4.2 Comparison

We compare the performance of two classifiers. One is our proposed RBFN-SLMP and the second is a Lattice Neural Network with Dendritic Processing (RNMPD). The metric used is the percentage of accuracy, defined as:

$$Acc = \frac{N_{correct}}{N_{Total}} \times 100$$

Table 1. Description of the benchmarks used.

Number of samples				
Databases	Features	Classes	Training	Test
LD	6	2	200	145
PIMA	8	2	400	368
BC	9	2	300	206
HEART	13	2	70	200
ION	34	2	100	251
IRIS	4	3	45	105
WINE	13	3	60	118

where $N_{correct}$ is the number of correct predictions and N_{Total} is the total number of data points.

Table 2, shows the results of the experiments. As it is possible to see, RBFN-SLMP outperforms RNMPD in all cases. In case that the number of features exceeded 13, we extracted new features using Principal Component Analysis (PCA), and we used the first 10 components. The test conditions are the same for both methods. Besides, we tested the algorithms with the databases standardized, and also using the raw data. We ran the experiments 20 times and calculated the average of the results.

Table 2. Comparison of the accuracy achieved by RBFN-SLMP and RNMPD.

	RBFN-SLMP	RNMPD
LD	68.73 %	57.59 %
PIMA	76.40 %	68.42 %
HEART	82.65 %	71.90 %
ION	91.62 %	85.88 %
BC	87.87 %	93.41 %
IRIS	95.33 %	89.81 %
WINE	94.58 %	86.44 %

Figure 2 shows how smaller values for sigma (the amplitude of the Gaussian) imply worse results for all databases. Also, it shows that changes in the number of iterations do not have a big impact in the performance.

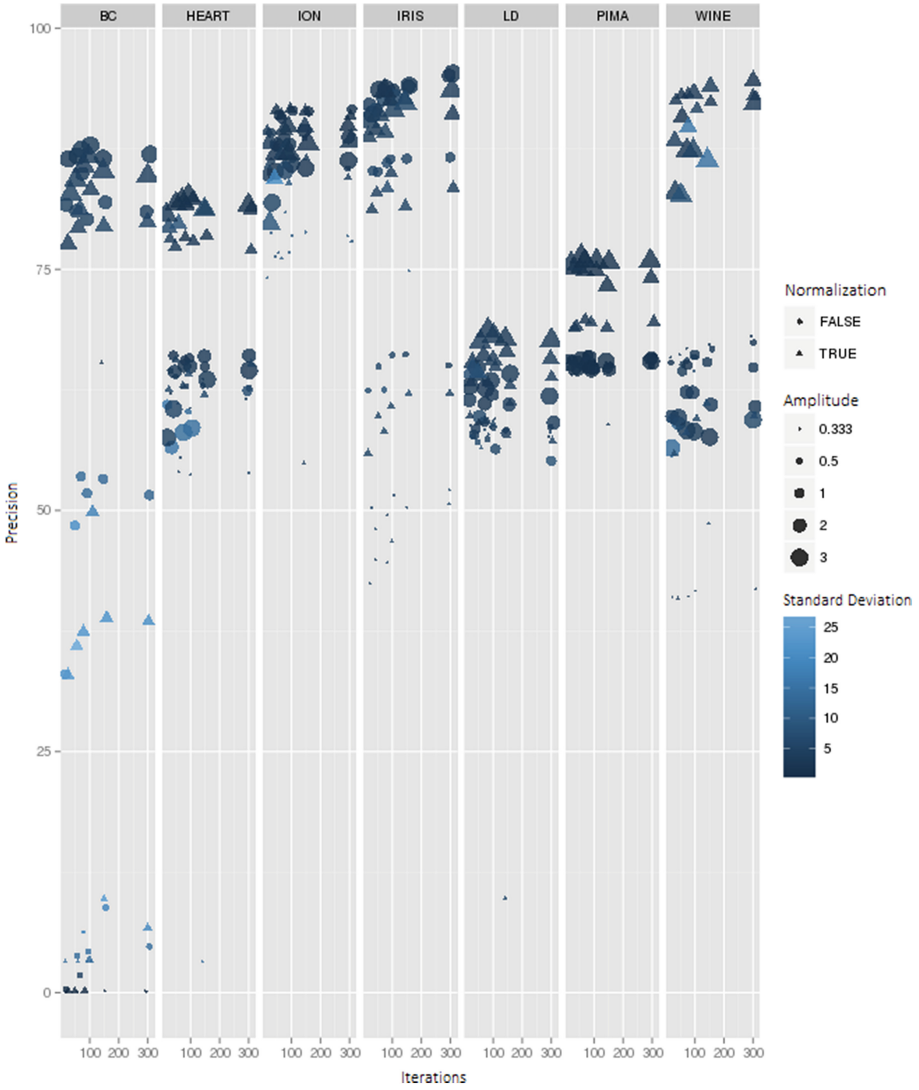

Fig. 2. The image shows the impact on the performance of the RBFNN, when modifying: iteration number, value of sigma and normalization.

5 Conclusion and Future Work

One of the main purposes of this paper was to test whether the algorithm that was originally developed to train a Lattice Neural Networks with Dendritic Processing (LNNDP) could be applied to the automatic selection of the parameters associated with the hidden layer of a Radial-Basis Function Neural Network (RBFNN). Also, it was our interest to compare the performance of the automatically tuned RBFNN with a LNNDP.

The results presented in Sect. 4 show that it is in fact possible to use the LNNDP training algorithm for the RBFNN. And they also offer data to support the claim that such a network shows a better performance than the LNNDP. According to the set of experiments that we have performed so far, the results show that the selection of the centers of the Gaussians allow a better accuracy in the classifier. Compared with the RNMDP, our approach always achieves a higher accuracy. Although it is not possible to generalize from only one set of experiments, we believe that the approach could be consider as an interesting option to alleviate the tuning of a radial-basis neural network.

Our contribution is to provide a method that automatically adjusts a RBFN depending on the data, requiring no intervention from the user, and giving a good accuracy, even better than the one given by some other classifiers. There are two main advantages in the method: (1) There is no intervention needed from the user. The method works automatically. (2) The amount and location of the radial-basis functions yields performances better or very similar to other approaches. The main disadvantage would be the limitation on the number of attributes that can be considered. Although some alternatives such as the use of Principal Component Analysis are possible.

Of course, there are a number of issues that could be explored in future work. We identify at least this ones:

- Calculate the amplitude σ_j for each hidden layer node.
- Include more classifications algorithms in the comparison, e.g. SVM, ELM, MLP, Bayes.
- Run a design experiment to see if iterations, sigma and normalization are statistically significant.
- In a extended version of this work, we consider studying the influence of parameters such as the amplitude of the Gaussians, and the number of hidden neurons for different problems.

Acknowledgments. The authors thank Tecnológico de Monterrey, Campus Guadalajara, as well as IPN-CIC under project SIP 20161126, and CONACYT under project 155014 and 65 within the framework of call: Frontiers of Science 2015 for the economical support to carry out this research.

References

1. Bishop, C.M., et al.: Pattern Recognition and Machine Learning. Information Science and Statistics, vol. 1. Springer, New York (2006)
2. Miller, A., Blott, B., et al.: Review of neural network applications in medical imaging and signal processing. Med. Biol. Eng. Comput. **30**(5), 449–464 (1992)
3. Willis, M., Montague, G., Di Massimo, C., Tham, M., Morris, A.: Artificial neural network based predictive control. In: Advanced Control of Chemical Processes (ADCHEM 1991): Selected Papers from the IFAC Symposium, p. 261, 14–16 October 1991. Toulouse, France (2014)
4. Le, T.H.: Applying artificial neural networks for face recognition. Adv. Artif. Neural Syst. **2011**, 16 (2011). doi:10.1155/2011/673016. Article ID 673016

5. Siniscalchi, S.M., Svendsen, T., Lee, C.H.: An artificial neural network approach to automatic speech processing. Neurocomputing **140**, 326–338 (2014)
6. Maren, A.J., Harston, C.T., Pap, R.M.: Handbook of Neural Computing Applications. Academic Press, New York (2014)
7. Haykin, S.: Neural Networks and Learning Machines. Number v. 10 in Neural networks and learning machines. Prentice Hall, Upper Saddle River (2009)
8. Park, J., Sandberg, I.W.: Universal approximation using radial-basis-function networks. Neural Comput. **3**(2), 246–257 (1991)
9. Chen, S., Cowan, C.F., Grant, P.M.: Orthogonal least squares learning algorithm for radial basis function networks. IEEE Trans. Neural Networks **2**(2), 302–309 (1991)
10. Moody, J., Darken, C.J.: Fast learning in networks of locally-tuned processing units. Neural Comput. **1**(2), 281–294 (1989)
11. Poggio, T., Girosi, F.: Networks for approximation and learning. Proc. IEEE **78**(9), 1481–1497 (1990)
12. Kurban, T., Beşdok, E.: A comparison of RBF neural network training algorithms for inertial sensor based terrain classification. Sensors **9**(8), 6312–6329 (2009)
13. Chun-Tao, M., Kun, W., Li-yong, Z.: A new training algorithm for RBF neural network based on PSO and simulation study. In: WRI World Congress on Computer Science and Information Engineering, vol. 4, pp. 641–645 (2009)
14. Oh, S.K., Kim, W.D., Pedrycz, W., Park, B.J.: Polynomial-based radial basis function neural networks (P-RBF NNs) realized with the aid of particle swarm optimization. Fuzzy Sets Syst. **163**(1), 54–77 (2011)
15. Reiner, P.D.: Algorithms for Optimal Construction and Training of Radial Basis Function Neural Networks. Ph.D. thesis, Auburn University (2015)
16. Sossa, H., Guevara, E.: Efficient training for dendrite morphological neural networks. Neurocomputing **131**, 132–142 (2014)
17. Ritter, G.X., Iancu, L., Urcid, G.: Morphological perceptrons with dendritic structure. In: Proceedings of the IEEE International Conference on Fuzzy Systems, vol. 2, pp. 1296–1301. IEEE (2003)
18. Fun, M.H., Hagan, M.T.: Recursive orthogonal least squares learning with automatic weight selection for Gaussian neural networks. In: Proceedings of the International Joint Conference on Neural Networks, vol. 3, pp. 1496–1500 (1999)
19. Bache, K., Lichman, M.: UCI machine learning repository (2013)
20. Babu, G.S., Suresh, S.: Sequential projection-based metacognitive learning in a radial basis function network for classification problems. IEEE Trans. Neural Netw. Learn. Syst. **24**(2), 194–206 (2013)