A Self-adaptive Growing Method for Training Compact RBF Networks

Baile Xu, Furao Shen(✉), Jinxi Zhao, and Tianyue Zhang

National Key Laboratory for Novel Software Technology,
Department of Computer Science and Technology,
Collaborative Innovation Center of Novel Software Technology and Industrialization,
Nanjing University, Nanjing, China
dg1633021@smail.nju.edu.cn, {frshen,jxzhao}@nju.edu.cn,
njucszty@gmail.com

Abstract. Radial Basis Function (RBF) network is a neural network model widely used for supervised learning tasks. The prediction time of a RBF network is proportional to the number of nodes in its hidden layer, while there is also a positive correlation between the number of nodes and the predication accuracy. In this paper, we propose a new training algorithm for RBF networks in order to construct high accuracy networks with as few nodes as possible. The proposed method starts with an empty network, selecting a best node from candidates iteratively until the training error reduces to a threshold or the number of nodes reaches a limit. Then the network is further optimized with a supervised fine-tuning method. Experimental results indicate that the proposed method could achieve better performances than traditional algorithms when training same sized RBF networks.

Keywords: Radial basis function networks · Nonlinear regression

1 Introduction

Radial Basis Function (RBF) network is a neural network model which has widely been used for supervised learning tasks, such as classification and function approximation, since it was proposed in late 1980s [1]. RBF network has attracted a lot of research interests due to its simple network architecture, good performance and solid theoretical foundation. The main idea of RBF network is approximating a nonlinear function with a linear combination of radial basis functions, which was first introduced as a solution of multivariable interpolation problems [2].

In general supervised learning tasks, training data can be learned perfectly by a RBF network if all data points are appointed as centers of radial basis functions. However, this would obviously cause overfitting, and the computation time for prediction would be unacceptably long because it is proportional to the number of radial basis functions. Therefore, an important task of training RBF

D. Liu et al. (Eds.): ICONIP 2017, Part I, LNCS 10634, pp. 74–81, 2017.
https://doi.org/10.1007/978-3-319-70087-8_8

networks is locating an appropriate number of RBF centers. In most existing training algorithms, the number of RBF centers need to be decided as a priority. A larger number of centers would boost the accuracy of the network at the cost of computation time for training and prediction. It would be beneficial if a training algorithm could automatically decide the number of RBF centers for specific tasks.

In this paper, we propose a new two step algorithm for training compact RBF networks, namely Growing Radial Basis Function Network(G-RBFN). It is composed of a network building step and a fine tuning step. Starting from an empty network, the proposed method iteratively selects best candidate nodes from training data points. Then the network is further optimized with supervised fine-tuning.

The rest of this paper is organized as follows. The architecture and training algorithms of RBF networks are introduced in Sect. 2. The proposed method is described in detail in Sect. 3. Section 4 presents experimental results. Section 5 concludes the paper. For the clarity of symbol usage, we use uppercase characters for denoting matrices, bold lowercase character for denoting vectors and normal lowercase characters for denoting numerical values respectively.

2 RBF Networks and Training Algorithms

The standard structure of RBF networks consists of an input layer, a hidden layer and an output layer. The networks take real-valued vectors as input, denoted as $\boldsymbol{x} \in R^n$. The hidden layer consists of a number of nodes whose activation functions are radial basis functions. Radial basis functions are a family of functions whose value only depend on the distance between the variable $\boldsymbol{x}$ and a center point $\boldsymbol{c}$, namely, a function ϕ that $\phi(\boldsymbol{x}, \boldsymbol{c}) = \psi(||\boldsymbol{x} - \boldsymbol{c}||)$. We assume that the output layer contains only one node for simplicity, and the network can be formulated into a function:

$$f(\boldsymbol{x}) = \sum_{j=1}^{k} w_i \phi(\boldsymbol{x}, \boldsymbol{c}_j) + w_0 \tag{1}$$

Here k is the number of nodes in hidden layer. The parameters in RBF networks consists of the output layer weights $\boldsymbol{w} = (w_0, w_1, w_2, ..., w_k)^T$, RBF centers $C = (\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_k)$ and intrinsic parameters of the radial basis function. The most commonly used radial basis function is Gaussian function with Euclidean distance:

$$\phi(\boldsymbol{x}, \boldsymbol{c}) = e^{-\frac{||\boldsymbol{x}-\boldsymbol{c}||^2}{2\sigma^2}} \tag{2}$$

The training data set is denoted as $(X, \boldsymbol{y}) = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_{|X|}, y_{|X|})\}$, where $|X|$ denotes the size of X. The task of training RBF networks is optimizing network parameters to minimize a loss function, which is usually a least square loss function $L(X, \boldsymbol{y}) = \sum_{i=1}^{|X|}(y_i - f(\boldsymbol{x}_i))^2$. The main problem is training hidden layer parameters especially C, because the output

layer weights $\boldsymbol{w}$ can be computed analytically once other parameters are confirmed. If the radial basis functions in the network are differentiable, then C can be trained by supervised gradient methods [3]. A problem of gradient methods is how to choose initial values in order to avoid local minimums. Moreover, gradient methods usually take a long time to converge. Another class of widely used methods is learning the centers by clustering training data. Clustering is usually faster than gradient methods, but as an unsupervised training method, it is difficult to achieve the optimal results. Some researches add supervision into the clustering procedure to improve its performance. For example, it is reported in [4] that class-specific clustering could raise the accuracy of RBF networks in classification tasks. Some training algorithms focus on learning the nodes in the hidden layer incrementally. These algorithms are mainly designed for online learning where training data points are learned sequentially, such as GGAP [5] and GGAP-GMM [6]. An offline training algorithm based on error correction was proposed in [7].

3 Growing Radial Basis Function Network

Growing Radial Basis Function Network(G-RBFN) can be summarized into two steps: a self-adaptive network building step and an error backpropagation based fine-tuning step. The network building step is a self-contained training algorithm itself, while the fine-tuning step could generally boost the accuracy of roughly trained RBF networks, especially the networks whose hidden layer parameters are trained by unsupervised methods. Gaussian function is used as the default radial basis function in G-RBFN, but the width parameter σ for each hidden node is different. In this paper the width parameters are denoted as $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, ..., \sigma_k)^T$, where σ_j denotes the width parameter of node j.

3.1 Network Building Step

The basic process of the network building step is iteratively selecting a candidate node with most potential contribution to be a new hidden layer node. The contribution of a node is defined as the error that removing the node would introduce. It can be seen from (1) that for a specific input $\boldsymbol{x}$, removing node j would introduce an error of $e_j(\boldsymbol{x}) = |w_j|\phi(\boldsymbol{x}, \boldsymbol{c}_j)$. Therefore if the density distribution of input data is already known as $p(\boldsymbol{x})$, then the total contribution of node j can be computed as:

$$e_j = \int |w_j|\phi(\boldsymbol{x}, \boldsymbol{c}_j)p(\boldsymbol{x})d\boldsymbol{x} \tag{3}$$

Generally speaking, e_j can not be computed by (3) in real world applications, because $p(\boldsymbol{x})$ is unknown as well as the integration is difficult to compute. Hence we use an approximation of e_j by assuming a discrete distribution over the

training data set X that $p(\boldsymbol{x} = \boldsymbol{x}_i) = \frac{1}{|X|}$, then the approximation can be computed as:

$$\hat{e}_j = \sum_{i=1}^{|X|} |w_j| \phi(\boldsymbol{x}_i, \boldsymbol{c}_j) \tag{4}$$

When a training data point $\boldsymbol{x}_i$ is considered as a candidate node, weight and width parameters have to be assigned to $\boldsymbol{x}_i$. The width parameter for candidate node is computed as:

$$\sigma_{\boldsymbol{x}_i} = min_j \sqrt{||\boldsymbol{x}_i - \boldsymbol{c}_j||} \tag{5}$$

The potential weight $w_{\boldsymbol{x}_i}$ is computed by eliminating the training error $L(X, \boldsymbol{y}) = \sum_{j=1}^{|X|}(y_j - f(\boldsymbol{x}_j))$ in current network. Hence we define the following regularized least square loss function, where λ is the regularization parameter:

$$l(w_{\boldsymbol{x}_i}) = \sum_{j=1}^{|X|}(y_j - f(\boldsymbol{x}_j) - w_{\boldsymbol{x}_i}\phi(\boldsymbol{x}_j, \boldsymbol{x}_i))^2 + \lambda w_{\boldsymbol{x}_i}^2 \tag{6}$$

The loss function is a convex function of $w_{\boldsymbol{x}_i}$, therefore the value of $w_{\boldsymbol{x}_i}$ that minimizes (6) can be computed as:

$$w_{\boldsymbol{x}_i} = \frac{\sum_{j=1}^{|X|}((y_j - f(\boldsymbol{x}_j))\phi(\boldsymbol{x}_j, \boldsymbol{x}_i))}{\sum_{j=1}^{|X|} \phi^2(\boldsymbol{x}_j, \boldsymbol{x}_i) + \lambda} \tag{7}$$

Then the contribution of the candidate node $\boldsymbol{x}_i$ can be computed by (4), and the candidate node with the most contribution is added to the hidden layer. A training data point is no longer considered as candidate node after it has been selected. When a new node is added, the parameters of old nodes need to be updated accordingly. The width parameters of old nodes are updated as:

$$\sigma_i = min_{j \neq i} \sqrt{||\boldsymbol{c}_i - \boldsymbol{c}_j||} \tag{8}$$

After the parameters of the hidden layer changed, the output weights $\boldsymbol{w}$ also needs updating. The loss function of $\boldsymbol{w}$ is defined as following:

$$l(\boldsymbol{w}) = \sum_{i=1}^{|X|}(y_i - f(\boldsymbol{x}_i))^2 + \lambda \boldsymbol{w}^T \boldsymbol{w} \tag{9}$$

Introducing a $|X| \times (k+1)$ matrix $\boldsymbol{\Phi}$ that:

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & \phi(\boldsymbol{x}_1, \boldsymbol{c}_1) & \cdots & \phi(\boldsymbol{x}_1, \boldsymbol{c}_k) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi(\boldsymbol{x}_{|X|}, \boldsymbol{c}_1) & \cdots & \phi(\boldsymbol{x}_{|X|}, \boldsymbol{c}_k) \end{bmatrix} \tag{10}$$

Then $\boldsymbol{w}$ is computed as:

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda I)^{-1} \boldsymbol{\Phi}^T \boldsymbol{y} \tag{11}$$

A learning round ends after updating $\boldsymbol{w}$. The network building step is simply repeating the learning rounds, starting from an empty network that $f(\boldsymbol{x}) = 0$ and ends when the root mean square error of training data falls under a predefined threshold $rmse_{min}$ or the number of nodes exceeds a limit k_{max}. Because the network is empty at the beginning, the width parameters of candidate nodes can not be computed by (5) at this time, therefore a default width σ_0 need to be set for all candidate nodes in the first iteration. We fix $\sigma_0 = 0.5$ because its effect is trivial.

3.2 Fine-Tuning Step

The function of G-RBFN is differentiable, therefore a supervised training with gradient methods can be used for fine-tuning the parameters of the network. In our implementation, stochastic gradient descent is adopted to train the network built in the previous step.

Unlike normal supervised training methods that train all parameters simultaneously, we only use stochastic gradient descent for training C and $\boldsymbol{\sigma}$. When a training data point $(\boldsymbol{x}_i, y_i)$ is learned, the parameters of node j are updated as following:

$$\Delta \boldsymbol{c}_j = \eta_1 w_j (y_i - f(\boldsymbol{x}_i)) \phi(\boldsymbol{x}_i, \boldsymbol{c}_j) (\frac{\boldsymbol{x}_i - \boldsymbol{c}_j}{\sigma_j^2}) \tag{12}$$

$$\Delta \sigma_j = \eta_2 w_j (y_i - f(\boldsymbol{x}_i)) \phi(\boldsymbol{x}_i, \boldsymbol{c}_j) (\frac{||\boldsymbol{x}_i - \boldsymbol{c}_j||^2}{\sigma_j^3}) \tag{13}$$

The stochastic gradient descent training iterates for a predefined number of epochs, and each training data point is learned once in a single epoch. The output weights $\boldsymbol{w}$ is computed by (11) at the end of each training epoch, in order to ensure $\boldsymbol{w}$ is fully converged.

Obviously, the supervised optimizing could be easily fused into the network building step. We could optimize the parameters of a new node with gradient descent after it is added, or apply the fine-tuning algorithm to the whole network at the end of each learning round. In this way the network is better optimized before adding new nodes, thus could be more compact. The reason for not doing so is that gradient based methods would make the training time considerably longer. We prefer to separate these time consuming computations into an optional optimizing step, thus the network building step could work as a self-contained RBF network training algorithm.

4 Experiments

4.1 Experiment on Artificial Data

In this section we illustrate the performance of G-RBFN on a low-dimensional function approximation problem. G-RBFN is applied to approximate a benchmark function: $z = cos(x)sin(y)$, where $x, y \in [0, 2\pi]$. The desired output is shown in Fig. 1.

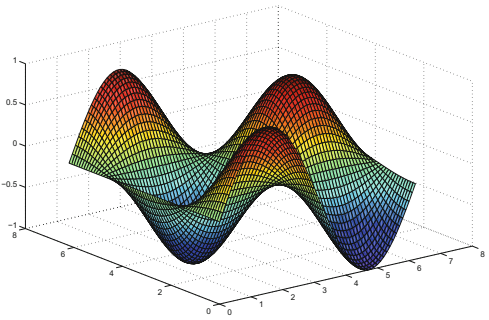
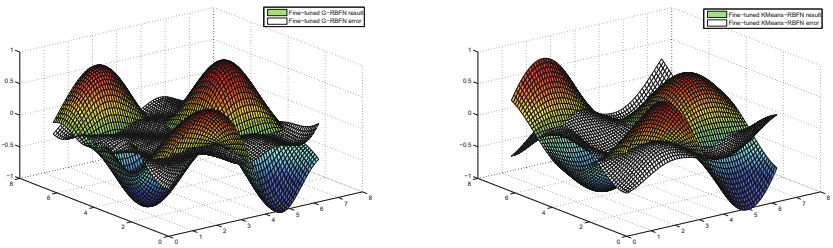


Fig. 1. Illustration of the benchmark function $z = cos(x)sin(y)$.

The training data set in this experiment consists of 4000 data points randomly sampled from the input space. The validation data set consists of 3969 points locating on corners of $0.1 * 0.1$ grids of the x-y plane, thus the validation results could be illustrated as a smooth curved surface. We compared G-RBFN with a RBF network denoted as KMeans-RBFN, whose hidden node centers are trained by KMeans clustering. The hyperparameters of G-RBFN network building step are set as: $rmse_{min} = 0.05, k_{max} = 100, \lambda = 0.01$. The number of hidden layer nodes in KMeans-RBFN is equal with that in G-RBFN. Both networks are fine-tuned with the same algorithm described in Sect. 3.2. The hyperparameters for fine-tuning are set as: $\eta_1 = 10^{-3}, \eta_2 = 10^{-5}, MaxEpoch = 100$. The results are illustrated in Fig. 2, where the colored surfaces illustrate the validation results and the white surfaces illustrate the residual errors. Approximation quality is evaluated by root mean square error($rmse$).



(a) G-RBFN $rmse = 0.0245$ (b) KMeans-RBFN $rmse = 0.0886$

Fig. 2. Results of benchmark function approximation. The colored surfaces illustrate the validation results and the white surfaces illustrate the residual errors. (Color figure online)

It can be seen from the figures that the approximation quality of G-RBFN is better than that of KMeans-RBFN. In this experiment, G-RBFN generated 25

hidden nodes before reducing training $rmse$ to 0.05. The validation error after network building is also near 0.05 after network building, which is significantly smaller than the error of KMeans-RBFN. Fine-tuning improved the accuracy of both networks. The rmse of G-RBFN is reduced from 0.0464 to 0.0245, and the rmse of KMeans-RBFN is reduced from 0.1180 to 0.0886.

4.2 Experiment on Real World Data

In this section we applied G-RBFN to many real world benchmark datasets for regression. All datasets are obtained from UCI machine learning repository [8], including Abalone, Airfoil, Auto MPG, Housing and Concrete Compressive Strength(CCS)[9]. Value ranges of data features and labels are all normalized to [0, 1]. We randomly divide each benchmark data set into a training set with 90% data and a test set with 10% data. All experiments are repeated 10 times and the average results are reported.

G-RBFN is compared with KMeans-RBFN and Multilayer Perceptron(MLP). The number of hidden layer nodes in KMeans-RBFN and MLP are set to be equal with that of G-RBFN. The hyperparameters of G-RBFN network building step are set as: $rmse_{min} = 0.08, k_{max} = 100, \lambda = 0.01$. The hyperparameters for fine-tuning are set as: $\eta_1 = 10^{-3}, \eta_2 = 10^{-5}, MaxEpoch = 100$. The results are reported in Table 1. The best result in each experiment is shown is bold symbol. For G-RBFN and KMeans-RBFN, the results before and after fine-tuning are both reported. The numbers in brackets are average results of fine-tuned networks.

Table 1. Experimental results on real-world data sets.

Data set	Nodes	G-RBFN rmse	KMeans-RBFN rmse	MLP rmse
Abalone	19.7	0.0770(**0.0735**)	0.0849(0.0739)	0.0739
Airfoil	100	0.0843(**0.0782**)	0.0878(0.0861)	0.0831
Auto-MPG	7.3	0.0739(0.0709)	0.0866(0.0777)	**0.0693**
CCS	92.4	0.0924(**0.0849**)	0.0987(0.0896)	0.0943
Housing	22	0.0948(**0.0898**)	0.1222(0.1056)	0.0940

It can be seen from Table 1 that fine-tuned G-RBFN outperformed KMeans-RBFN and Multilayer Perceptron in most experiments. The accuracy of G-RBFN without fine-tuning is also acceptable, which is comparable with fine-tuned KMeans-RBFN. The number of nodes in each experiment are also reported. It is observable that the number of nodes in different experiments varies largely, mainly because some data sets are more difficult to learn than others. Although the number of nodes in G-RBFN is automatically decided, it is still controllable by adjusting the training error threshold. We recommend setting a small training error threshold like $rmse_{min} = 0.05$ at the beginning, and gradually raising the threshold until the number of nodes falls below k_{max}.

5 Conclusion

A growing network model named G-RBFN for training radial basis function network is proposed in this paper. G-RBFN consists of a network building step and a fine-tuning step. The network building step repeatedly add a new hidden layer node that has the largest potential contribution to the network until the training error reduces to a threshold or the number of nodes exceeds a limit. The fine-tuning step further optimizes the network parameters with gradient based supervised training. Experimental results on artificial and real-world data prove that G-RBFN performs better than traditional training algorithms for RBF networks.

However, G-RBFN still has room for improvement. The fine-tuning step only uses the standard supervised training method without applying any training tricks. The network compactness could be further improved by adding pruning methods to remove less effective nodes. G-RBFN also has the potential to be modified into an online learning algorithm, which would be an interesting topic for future research.

Acknowledgments. This work is supported in part by the National Science Foundation of China under Grant Nos. (61373130, 61375064, 61373001), and Jiangsu NSF grant (BK20141319).

References

1. Broomhead, D.S., Lowe, D.: Radial basis functions, multi-variable functional interpolation and adaptive networks. In: Advances in Neural Information Processing Systems RSRE-memo-4148, pp. 728–734 (1988)
2. Powell, M.J.D.: Radial basis functions for multivariable interpolation: a review. In: Algorithms for Approximation, pp. 143–167 (1987)
3. Werbos, P.J.: Backpropagation: past and future. In: IEEE International Conference on Neural Networks IEEE, vol. 1, pp. 343–353 (1988)
4. Raitoharju, J., Kiranyaz, S., Gabbouj, M.: Training radial basis function neural networks for classification via class-specific clustering. IEEE Trans. Neural Networks Learn. Syst. **99**, 1–14 (2015)
5. Huang, G.B., Saratchandran, P., Sundararajan, N.: A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. IEEE Trans. Neural Networks **16**(1), 57–67 (2005)
6. Bortman, M., Aladjem, M.: A growing and pruning method for radial basis function networks. IEEE Trans. Neural Networks **20**(6), 1039–1045 (2009)
7. Yu, H., et al.: An incremental design of radial basis function networks. IEEE Trans. Neural Networks Learn. Syst. **25**(10), 1793–1803 (2014)
8. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
9. Yeh, I.C.: Modeling of strength of high-performance concrete using artificial neural networks. Cement Concrete Res. **28**(12), 1797–1808 (1998)