AN EFFICIENT LEARNING ALGORITHM FOR FUNCTION APPROXIMATION WITH RADIAL BASIS FUNCTION NETWORKS

Yen-Jen Oyang and Shien-Ching Hwang

Department of Computer Science and Information Engineering

National Taiwan University, Taipei, Taiwan

yjoyang@csie.ntu.edu.tw, schwang@mars.csie.ntu.edu.tw

ABSTRACT

This paper proposes a novel learning algorithm for constructing function approximators with radial basis function (RBF) networks. In comparison with the existing learning algorithms, the proposed algorithm features lower time complexity for constructing the RBF network and is able to deliver the same level of accuracy. The time taken by the proposed algorithm to construct the RBF network is in the order of $O(|S|)$, where S is the set of training samples. As far as the time complexity for predicting the function values of input vectors is concerned, the RBF network constructed with the proposed learning algorithm can complete the task in $O(|T|)$, where T is the set of input vectors. Another important feature of the proposed learning algorithm is that the space complexity of the RBF network constructed is $O(m|S|)$, where m is the dimension of the vector space in which the target function is defined.

Key terms: Radial basis function network, Function approximation, Machine learning.

1. INTRODUCTION

Function approximation is one of the fundamental issues in machine learning [5] and has been exploited by computer scientists in a number of applications, including data classification and data mining [3]. One widely-used form of function approximators is a linear combination of spherical Gaussian functions as shown in the following:

$$h(x) = w_0 + \sum_{j=1}^n w_j \exp\left(-\frac{\|x - c_j\|^2}{2\sigma_j^2}\right) \quad (1)$$

where c_j 's are the centers of the Gaussian functions, and $\|x - c_j\|$ is the distance between vector x and c_j . Hartman et al. have shown that the linear combination form shown in equation (1) can approximate any function with arbitrarily small error [1].

A function approximator based on equation (1) is generally regarded as a radial basis function (RBF) network. The task of the learning algorithm is to determine the centers, weights, and variances of the Gaussian basis functions. In the original proposal of regularization RBF [11], the number of basis functions equals the number of training samples. The basis

functions are centered on the training samples and the only unknown parameters are the weights, which can be determined by solving a system of linear equations. However, the resulting networks are complex and often ill-conditioned. Generalized RBF networks are designed with fewer nodes than the number of samples in the training set. As a result, the networks constructed are less complex. The main challenge in such an approach is to determine where the basis functions should be placed and the weights and variances of these basis functions. Several different strategies have been proposed and a good survey can be found in [2]. In recent years, M. Orr proposed a number of alternative learning algorithms for constructing function approximators with RBF networks [6, 7, 8, 9]. However, the algorithms proposed by Orr suffer high time complexities in the construction of the RBF network, due to the need to compute the inverse of a matrix.

In this paper, an efficient learning algorithm for constructing RBF networks based function approximators is proposed. The time complexity for constructing a RBF network with the proposed learning algorithm is $O(|S|)$, where S is the set of training samples. As far as the time complexity for predicting the function values of input vectors is concerned, the RBF network constructed can complete the task in $O(|T|)$, where T is the set of input vectors. Experimental results reveal that the RBF network constructed with the proposed learning algorithm is able to approximate the target function at the same level of accuracy as the RBF networks constructed with the learning algorithms proposed by M. Orr [6, 7, 8, 9]. Another important feature of the proposed learning algorithm is that the space complexity of the RBF network constructed is $O(m|S|)$, where m is the dimension of the vector space in which the target function is defined.

In the following part of this paper, section 2 elaborates the proposed learning algorithm. Section 3 reports the experiments conducted to evaluate the proposed learning algorithm. Finally, concluding remarks are presented in section 4.

2. THE PROPOSED LEARNING ALGORITHM

The learning algorithm proposed in this paper assumes that samples are taken uniformly over the space of the

function. If the training samples are not exactly located at the crosses of the gridlines, then some sort of interpolation should be conducted. The learning algorithm places one spherical Gaussian function at each training sample and determines the appropriate weights and variances for these Gaussian basis functions. Let f be a function defined in an m -dimensional vector space. Assume that samples are taken uniformly at $(b_1\delta, b_2\delta, \dots, b_m\delta)$ over the space of function f , where $b_1, b_2, \dots, b_m$ are integers. The learning algorithm presented in this paper will construct a function approximator of the following form:

$$\hat{f}(x) = \sum_{s_i \in S} w_i \exp\left(-\frac{\|x - s_i\|^2}{2\sigma_i^2}\right) \quad (2)$$

where S is the set of training samples, w_i and σ_i^2 are the weight and variance of the Gaussian function centered at sample s_i , respectively. In the following discussion, it is first assumed that the data set is noiseless. How to deal with noisy data sets will be elaborated later.

Since the influence of a Gaussian function decreases exponentially as the distance increases, when computing the function value at x_0 based on equation (2), we can include only the Gaussian functions in the proximity of x_0 . In other words, we only need to include the Gaussian functions located within a sphere centered at x_0 . The problem now is how large the sphere should be. In the proposed learning algorithm, the variances of the Gaussian basis functions are uniformly set to a multiple of δ . Therefore, regardless of where x_0 is located, the radius of the sphere can be set to a fixed real number determined by the value of δ . Furthermore, because the training samples are uniformly distributed, we can, in practice, simply include a fixed number of the Gaussian functions centered at the nearest training samples of x_0 , regardless of where x_0 is located.

Another assumption that the proposed learning algorithm makes is that the density of training samples is sufficiently high so that the values of function f at the nearby samples of a sample s_i are all virtually equal to $f(s_i)$. One may argue that this assumption may not hold in real cases. However, the weights of the Gaussian functions determined based on this assumption are just initial guesses. An iterative process will be conducted to fine-tune the weights. Based on this assumption, we can expect that the learning algorithm will set the weights and variances of the Gaussian basis functions centered at s_i and the nearby samples to virtually the same values. Therefore, the problem of figuring out the weight and variance of the Gaussian basis function centered at s_i can be modeled by figuring out w_i and σ_i such that

$$w_i \exp\left(-\frac{\|x - s_i\|^2}{2\sigma_i^2}\right) + \sum_{j=1}^{k-1} w_i \exp\left(-\frac{\|x - v_j\|^2}{2\sigma_i^2}\right) = f(s_i) \quad (3)$$

where x is a vector in the proximity of s_i and $v_1, v_2, \dots, v_{k-1}$ are the $k-1$ nearest samples surrounding s_i .

In order to simplify the subsequent discussion, we assume that function f is shifted so that s_i is located at the origin. As a result, we can rewrite equation (3) as

$$w_i \left[\exp\left(-\frac{\|x\|^2}{2\sigma_i^2}\right) + \sum_{j=1}^{k-1} \exp\left(-\frac{\|x - v_j\|^2}{2\sigma_i^2}\right) \right] = f(s_i). \quad (4)$$

$$\text{Let } g(x) = w_i \left[\sum_{j_1=-\infty}^{\infty} \dots \sum_{j_m=-\infty}^{\infty} \exp\left(-\frac{\|x - (j_1\delta, \dots, j_m\delta)\|^2}{2\sigma_i^2}\right) \right].$$

Then, we have

$$g(x) \equiv w_i \left[\exp\left(-\frac{\|x\|^2}{2\sigma_i^2}\right) + \sum_{j=1}^{k-1} \exp\left(-\frac{\|x - v_j\|^2}{2\sigma_i^2}\right) \right], \quad (5)$$

because the influence of a Gaussian function decreases exponentially as the distance increases. If we take partial derivatives of $g(x_1, x_2, \dots, x_m)$, then we will find that

$$\frac{\partial g(0, 0, \dots, 0)}{\partial x_j} = 0 \quad \text{and} \quad \frac{\partial g(\pm \frac{\delta}{2}, \pm \frac{\delta}{2}, \dots, \pm \frac{\delta}{2})}{\partial x_j} = 0, \quad \text{for}$$

all $1 \leq j \leq m$. In fact, it can be proved in mathematics that local maximums and local minimums of function g are located at $(a_1\delta, a_2\delta, \dots, a_m\delta)$ and $((b_1 + \frac{1}{2})\delta, (b_2 + \frac{1}{2})\delta, \dots, (b_m + \frac{1}{2})\delta)$, respectively, where $a_1, a_2, \dots, a_m$ and $b_1, b_2, \dots, b_m$ are integers. Therefore, one approach to make equation (4) above satisfied is to find w_i, σ_i such that

$$g(0, 0, \dots, 0) = g(\pm \frac{\delta}{2}, \pm \frac{\delta}{2}, \dots, \pm \frac{\delta}{2}) = f(s_i). \quad (6)$$

We have

$$\begin{aligned} g(0, 0, \dots, 0) &= w_i \left[\sum_{j_1=-\infty}^{\infty} \dots \sum_{j_m=-\infty}^{\infty} \exp\left(-\frac{(j_1^2 + \dots + j_m^2)\delta^2}{2\sigma_i^2}\right) \right] \\ &= w_i \left[\left(\sum_{j=-\infty}^{\infty} \exp\left(-\frac{j^2\delta^2}{2\sigma_i^2}\right) \right) \right]^m \end{aligned}$$

and

$$\begin{aligned} g(\pm \frac{\delta}{2}, \pm \frac{\delta}{2}, \dots, \pm \frac{\delta}{2}) &= w_i \left[\sum_{j_1=-\infty}^{\infty} \dots \sum_{j_m=-\infty}^{\infty} \exp\left(-\frac{[(j_1 + \frac{1}{2})^2 + \dots + (j_m + \frac{1}{2})^2]\delta^2}{2\sigma_i^2}\right) \right] \\ &= w_i \left[\left(\sum_{j=-\infty}^{\infty} \exp\left(-\frac{(j + \frac{1}{2})^2\delta^2}{2\sigma_i^2}\right) \right) \right]^m. \end{aligned}$$

To make equation (6) satisfied, we need to set σ_i so that

$$\sum_{j=-\infty}^{\infty} \exp\left(-\frac{j^2\delta^2}{2\sigma_i^2}\right) = \sum_{j=-\infty}^{\infty} \exp\left(-\frac{(j + \frac{1}{2})^2\delta^2}{2\sigma_i^2}\right) \quad (7)$$

Let $z = \exp\left(-\frac{\delta^2}{8\sigma_i^2}\right)$. Then, we can rewrite equation

(7) as

$$1 + 2 \sum_{j=1}^{\infty} z^{(2j)^2} = 2 \sum_{j=0}^{\infty} z^{(2j+1)^2}. \quad (8)$$

Since $0 < z < 1$, if we want to find the root of equation (8), we can take a limited number of terms from both sides of the equation and find the numerical roots. For

example, if we take the first 5 terms from each side of equation (8), then we have

$$1 + 2 \sum_{j=1}^4 z^{(2j)^2} = 2 \sum_{j=0}^4 z^{(2j+1)^2} \quad (9)$$

The only numerical real root of equation (9) is 0.861185. Let $z_0 = 0.861185$. Then, we have

$$\begin{aligned} & \left(1 + 2 \sum_{j=1}^{\infty} z_0^{(2j)^2} \right) - \left(2 \sum_{j=0}^{\infty} z_0^{(2j+1)^2} \right) \\ &= 1 + 2 \sum_{j=1}^4 z_0^{(2j)^2} - 2 \sum_{j=0}^4 z_0^{(2j+1)^2} + 2 \sum_{j=5}^{\infty} (z_0^{(2j)^2} - z_0^{(2j+1)^2}) \\ &= 2 \sum_{j=5}^{\infty} (z_0^{(2j)^2} - z_0^{(2j+1)^2}) > 0 \end{aligned}$$

Furthermore, we have

$$\begin{aligned} 2 \sum_{j=5}^{\infty} (z_0^{(2j)^2} - z_0^{(2j+1)^2}) &= 2z_0^{100} - 2 \sum_{j=5}^{\infty} (z_0^{(2j+1)^2} - z_0^{(2j+2)^2}) \\ &< 2z_0^{100} = 6.4664 \times 10^{-7}. \end{aligned}$$

In summary, we have

$$0 < \left(1 + 2 \sum_{j=1}^{\infty} z_0^{(2j)^2} \right) - \left(2 \sum_{j=0}^{\infty} z_0^{(2j+1)^2} \right) < 6.4664 \times 10^{-7}.$$

That is, $z_0 = 0.861185$ is a good approximate root of equation (8).

In fact, we can find other approximate roots of equation (8) by taking more or less terms from the two sides of the equation. Tab. 1 lists the other approximate roots and the corresponding error bounds. As Tab. 1 reveals, if more terms in equation (8) are included, then the upper bound of error is smaller. However, the magnitude of error is not the only concern for selecting an approximate root. We will address another effect in later discussion.

Number of terms taken from each side of equation (8)	Numerical root	Error bound
6	0.882248	2.8013×10^{-8}
7	0.897739	1.2580×10^{-9}
8	0.909615	5.6212×10^{-11}
9	0.919010	2.5010×10^{-12}

Tab. 1: Approximate roots of equation (8) and the corresponding error bounds.

Given that $z = \exp\left(\frac{-\delta^2}{8\sigma_i^2}\right)$ and that $z_0 = 0.861185$

is a good approximate root of equation (8), we therefore want to set σ_i as follows in order to make equation (7) satisfied:

$$\sigma_i = \sqrt{\frac{1}{8 \ln \frac{1}{z_0}}} \cdot \delta = 0.91456 \cdot \delta. \quad (10)$$

Equation (10) implies that we want to set the variances

of the Gaussian basis functions in the function approximator uniformly to $\frac{\delta^2}{8 \ln \frac{1}{z_0}}$.

With the variances of the Gaussian basis functions determined, the next issue to solve is to determine the weights of these Gaussian functions. Based on equation (3), we can derive one equation with each training sample. As a result, if we have n training samples, then we have n equations with n variables, which are the weights of the Gaussian functions. Accordingly, we can solve these equations and find the appropriate weights. However, the time complexity for solving n equations with n variables is very high, exceeding $O(n^2)$. Therefore, the empirical approach elaborated in the following is employed in this paper.

Based on equation (6), the empirical approach begins with setting

$$w_i = \frac{f(s_i)}{\left(1 + 2 \sum_{j=1}^{\infty} \exp\left(-\frac{j^2}{2\beta^2}\right) \right)^m},$$

where $\beta = \frac{\sigma_i}{\delta} = \sqrt{\frac{1}{8 \ln \frac{1}{z_0}}}$. Accordingly, the RBF

network based function approximator constructed by the learning algorithm is

$$\hat{f}(x) = \frac{1}{\lambda} \sum_{j=1}^k f(v_j) \exp\left(-\frac{\|x - v_j\|^2}{2\beta^2 \delta^2}\right), \quad (11)$$

where $\lambda = \left[1 + 2 \sum_{j=1}^{\infty} \exp\left(-\frac{j^2}{2\beta^2}\right) \right]^m$, x is a vector in the

space of f , and $v_1, v_2, \dots, v_k$ are the k nearest training samples of x . In fact, when evaluating λ , we only need to include a limited number of the exponential

terms, because $0 < e^{-\frac{j^2}{2\beta^2}} < 1$. With $z_0 = 0.861185$, we have $\beta = 0.91456$ and $\lambda \equiv (2.2925)^m$.

Fig. 1 uses an example to demonstrate the effect of the learning algorithm based on equation (11). As Fig. 1(a) shows, the function approximator based on equation (11) deviates from the target function by certain degree around the local minimum and local maximum of the function. The deviation is due to the assumption that the sampling density is sufficiently high. Since the approximate function value computed based on equation (11) is actually a weighted average of the training samples, the smoothing effect shown in Fig. 1(a) is expected. As Fig. 1(b) shows, if the density of samples is higher, then the deviation is smaller. In this paper, an iterative procedure is developed to compensate the smoothing effect. As shown in Fig. 1(c), after the compensation procedure has been conducted, the curve of the function approximator matches that of the target function more accurately. Details of the compensation procedure will be elaborated in the later part of this paper, when noise reduction is addressed.

Fig. 1(a). The smoothing effect of the function approximator based on equation (11), when the density of samples is low.

Fig. 1(b). The smoothing effect of the function approximator, when the density of samples is high.

Fig. 1(c). Effect of the compensation procedure.

Now, let us consider another factor to consider in selecting an approximate root of equation (8) from Table 1. If a larger approximate root is selected, then the global variance of the Gaussian basis functions will be larger. As a result, the smoothing effect will be more serious and more iterations in the compensation process need to be conducted.

The discussion so far assumes that the data set is noiseless. In most real-world applications, this ideal assumption does not hold. Therefore, a noise reduction mechanism must be incorporated. In this paper, a naïve low-pass filter [12] is incorporated at the input of the learning algorithm. Let s_i be a training sample, and $v_1, v_2, \dots, v_q$ be the q nearest samples surrounding s_i , and $\varepsilon(s_i), \varepsilon(v_1), \dots, \varepsilon(v_q)$ be the random noises associated

with these training samples, respectively. The learning algorithm will use $\bar{f}(s_i)$ defined as follows, instead of the observed $f(s_i) + \varepsilon(s_i)$, as the input at sample s_i :

$$\begin{aligned} \bar{f}(s_i) &= \frac{1}{q+1} \left[(f(s_i) + \varepsilon(s_i)) + \sum_{j=1}^q (f(v_j) + \varepsilon(v_j)) \right] \\ &= \frac{1}{q+1} \left[f(s_i) + \sum_{j=1}^q f(v_j) \right] + \frac{1}{q+1} \left[\varepsilon(s_i) + \sum_{j=1}^q \varepsilon(v_j) \right] \end{aligned} \quad (12)$$

At the output of the low-pass filter, the variance of noise is reduced by a factor of $q+1$, because

$$\text{Var} \left[\frac{\varepsilon(s_i) + \varepsilon(v_1) + \dots + \varepsilon(v_q)}{q+1} \right] = \frac{\text{Var}[\varepsilon(s_i)]}{q+1}.$$

With the low-pass filter incorporated, the function approximator based on equation (11) becomes:

$$\hat{f}(x) = \frac{1}{\lambda} \sum_{j=1}^k \bar{f}(v_j) \exp \left(-\frac{\|x - v_j\|^2}{2\beta^2\delta^2} \right). \quad (13)$$

However, the removal of high-frequency components with the low-pass filter is indiscriminate and results in a smoothing effect on the profile of the function. The learning algorithm then resorts to an iterative procedure to compensate the smoothing effect. The compensation procedure will fine-tune the weights of the Gaussian basis functions in equation (13) based on the assumption that $E[\varepsilon(t)] = 0$ for every training sample t , where $E[\varepsilon(t)]$ denotes the expected value of random variable $\varepsilon(t)$. According to the law of large numbers in Probability theory, for any real number $\xi > 0$, we have

$$\text{Prob} \left(\left| \frac{\varepsilon(s_i) + \varepsilon(v_1) + \dots + \varepsilon(v_r)}{r+1} \right| > \xi \right) \rightarrow 0, \text{ as } r \rightarrow \infty.$$

In other words, if r is sufficiently large, then it is highly likely that $\frac{\varepsilon(s_i) + \varepsilon(v_1) + \dots + \varepsilon(v_r)}{r+1} \approx 0$. Therefore, if

$$\hat{f}(s_i) \equiv f(s_i), \quad \hat{f}(v_1) \equiv f(v_1), \quad \hat{f}(v_2) \equiv f(v_2), \quad \dots, \quad \hat{f}(v_r) \equiv f(v_r),$$

$$\text{then we have} \quad \frac{1}{r+1} \left[(f(s_i) + \varepsilon(s_i)) + \sum_{j=1}^r (f(v_j) + \varepsilon(v_j)) \right]$$

$$\begin{aligned} &= \frac{1}{r+1} \left[\hat{f}(s_i) + \sum_{j=1}^r \hat{f}(v_j) \right] \\ &= \frac{1}{r+1} \left[(f(s_i) - \hat{f}(s_i)) + \sum_{j=1}^r (f(v_j) - \hat{f}(v_j)) \right] \\ &\quad + \frac{1}{r+1} \left[\varepsilon(s_i) + \sum_{j=1}^r \varepsilon(v_j) \right] \equiv 0 \end{aligned}$$

What the compensation procedure performs is adjusting $\hat{f}(s_i), \hat{f}(v_1), \dots, \hat{f}(v_r)$ so that

$$\begin{aligned} &\frac{1}{r+1} \left[(f(s_i) - \hat{f}(s_i)) + \sum_{j=1}^r (f(v_j) - \hat{f}(v_j)) \right] \\ &\quad + \frac{1}{r+1} \left[\varepsilon(s_i) + \sum_{j=1}^r \varepsilon(v_j) \right] \equiv 0. \end{aligned} \quad (14)$$

The pseudo-code of the compensation procedure is as follows:

```

procedure compensation(  $\hat{f}(s_1), \hat{f}(s_2), \dots, \hat{f}(s_n)$  );
 $s_1, s_2, \dots, s_n$ : training samples;
 $\eta$ : a positive real number  $\ll 1$ ;
for  $j = 1$  to  $\tau$  do {
  for  $i = 1$  to  $n$  {
    let  $v_1, v_2, \dots, v_r$  be the  $r$  nearest training samples of  $s_i$ ;
    
$$\Delta = \frac{1}{r+1} \left[ (f(s_i) + \varepsilon(s_i)) + \sum_{j=1}^r (f(v_j) + \varepsilon(v_j)) \right]$$

    
$$- \frac{1}{r+1} \left[ \hat{f}(s_i) + \sum_{j=1}^r \hat{f}(v_j) \right];$$

    
$$\hat{f}(s_i) = \hat{f}(s_i) + \eta \cdot \frac{\Delta}{\lambda};$$

  }
}

```

The final form of the function approximator is as follows:

$$\hat{f}(x) = \frac{1}{\lambda} \sum_{j=1}^k \hat{f}(v_j) \exp\left(-\frac{\|x - v_j\|^2}{2\beta^2\delta^2}\right), \quad (15)$$

where x is a vector in the space of f , $v_1, v_2, \dots, v_k$ are the k nearest training samples of x , and β and λ are the same constants defined in equation (11).

As far as the time complexity of the learning algorithm is concerned, there are four major tasks carried out by the learning algorithm in the construction of a RBF network. The first task is to construct a hypercube data structure to store the training samples. This task can be done in $O(|S|)$, where S is the set of training samples. The second task is to conduct low-pass filtering at the input of the function approximator. The time complexity of this task is $O(q|S|)$, where q is number of neighboring samples involved in equation (12). The third task is to construct a function approximator based on equation (13). The only work involved in this task is to determine the values of β and λ , which can be done in constant time. The last task that the learning algorithm carries out is executing the compensation procedure. This task can be done in $O(\tau|S|)$, where τ is the number of iterations of the outer loop of the compensation procedure. As a result, the overall time complexity of the learning algorithm in the construction of a RBF network is $O(|S| + q|S| + \tau|S|)$. If q , τ , and r are treated as constants, then the time complexity is in a linear order.

With equation (15), the time complexity of the function approximator for predicting the function values of input vectors is $O(k) + O(k|T|)$, where T is the set of input vectors and k is the number of nearest training samples included in equation (15). The term $O(k)$ is due to the time taken to figure out the k nearest training samples of an incoming vector. Because the training samples are evenly spaced, this task can be done in $O(k)$. If k is treated as a constant, then the time complexity is

$O(|T|)$. The space complexity of the RBF network is $O(m|S|)$, where m is the dimension of the vector space in which the target function is defined, because the number of nodes in the hidden layer of the RBF network equals to the number of training samples.

3. EXPERIMENTAL RESULTS

This section reports the experiments conducted to evaluate the proposed learning algorithm. The following three functions are used in the experiments:

- (i) $f_1(x) = 1 + (1 - x + 2x^2)e^{-x^2}$, $x \in [-5, 5]$;
- (ii) $f_2(x) = e^{-|x|}$, $x \in [-4, 4]$;
- (iii) $f_3(x_1, x_2) = 0.8 \cos(0.25x_1) \sin(0.5x_2)$,
 $x_1 \in [-2, 6]$, $x_2 \in [1, 10]$.

The first two functions are 1-dimensional, while the third function is 2-dimensional. The first function is the well-known Hermite polynomial [4]. In each experiment, 100 uniformly distributed samples are taken. A normally distributed random noise with variance = 0.1 is added to each training sample of f_1 and f_2 , while a normally distributed random noise with variance = 0.03 is added to each training sample of f_3 . There is no particular reason why different variances are used, except that it is of interest to observe the impact of different values.

Tab. 2 shows how the parameters of the proposed learning algorithm are set in these experiments. Parameter k is not set, because the number of training samples is not large. As a result, all the basis functions will be included in equation (15). Parameter q is set to 5, because a smaller number does not produce significant effect. A reasonable range of parameter q may be from 5 to 10. A value much larger than this range is not recommended, because the smoothing effect will be more serious. Our experiences suggest that parameter r should be larger than parameter q . Otherwise, overfitting may occur. The setting of parameter τ and η are also based on our experience.

The quality of the function approximators is evaluated based on root mean square error (RMSE) defined as follows:

$$RMSE = \sqrt{\frac{\sum_{x \in T} [f(x) - \hat{f}(x)]^2}{|T|}},$$

where T is the set of test samples. In each experiment, the test data set contains 100 randomly selected noiseless samples.

Parameter	Value
k in equation (15)	-
q in equation (12)	5
r in equation (14)	9
τ in compensation procedure	5
η in compensation procedure	0.5
z_0 : the numerical root of equation (8)	0.919010

Table 2. The parameter setting in the proposed learning algorithm.

In these experiments, the proposed learning algorithm is compared with the three approaches proposed by Orr [6, 7, 9]. The three Matlab functions **rbf_fs_2**, **rbf_rr_2**, and **rbf_rt_2** that implement Orr's approaches are available on [10]. How the parameters and options are set in these modules is listed in Table 3. The list is not exhaustive. For the parameters or options not included in Table 3, the default values were used. The 4 scales used in the experiments are those used in [10].

Parameter or option	Setting
Type of RBF	Gaussian
Model selection criterion	Bayesian information criterion
Bias	true
Scales	[1, 0.5, 0.2, 0.1]

Table 3. The parameter/option settings in **rbf_fs_2**, **rbf_rr_2**, and **rbf_rt_2**.

Table 4 shows the *RMSEs* observed in the experiments. The first number in each entry is the average of 100 independent runs and the second number is the standard deviation. In each of the 100 independent runs, a separate set of training samples is generated. As the numbers in Table 4 show, the proposed learning algorithm delivers the same level of accuracy as the three approaches proposed by Orr in these experiments.

	f_1	f_2	f_3
rbf_fs_2	0.036±0.008	0.025±0.002	0.090±0.022
rbf_rr_2	0.040±0.008	0.021±0.002	0.122±0.024
rbf_rt_2	0.045±0.012	0.016±0.003	0.097±0.007
proposed	0.040±0.007	0.016±0.002	0.065±0.005

Table 4. Experimental results measured by RMSE in 100 independent runs.

4. CONCLUSIONS

In this paper, an efficient learning algorithm for constructing function approximators with RBF networks is proposed. The proposed learning algorithm features a linear time complexity for constructing the RBF network. On the other hand, the most well-known existing learning algorithms suffer high time complexity due to the need to compute the inverse of a matrix. Experimental results reveal that the RBF network constructed with the proposed learning algorithm is able to approximate the target function at the same level of accuracy as the RBF networks constructed with the existing learning algorithms. As far as the time complexity for predicting the function values of input vectors is concerned, the RBF network constructed with the proposed learning algorithm can complete the task in $O(|T|)$, where T is the set of input vectors. Another important feature of the proposed learning algorithm is that the space complexity of the RBF network

constructed is $O(m|S|)$, where m is the dimension of the vector space in which the target function is defined.

Several issues deserve further studies. The first issue is data reduction. The current version of the proposed learning algorithm places one basis function at each training sample. It is of interest to develop some kind of data reduction mechanisms to reduce the complexity of the RBF network constructed. The second issue concerns noise reduction. In this paper, an empirical approach is incorporated. Since noise handling is a critical and challenging issue, further studies are highly desirable. The third issue is the general characteristics of the proposed learning algorithm. More comprehensive experiments should be conducted to learn more about its general characteristics.

5. REFERENCES

- [1] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with Gaussian hidden units as universal approximations," *Neural Computation*, vol. 2, no. 2, pp. 210-215, 1990.
- [2] S. Haykin, "Neural networks: A comprehensive foundation," New York: McMillan, 1994.
- [3] V. Kecman, *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, The MIT Press, Cambridge, Massachusetts, London, England, 2001.
- [4] D. J. C. Mackay, "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415-447, 1992.
- [5] T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
- [6] M. J. L. Orr, "Regularisation in the selection of radial basis function centres," *Neural Computation*, vol. 7, no. 3, pp. 606-623, 1995.
- [7] M. J. L. Orr, "Introduction to radial basis function networks," Technical report, Center for Cognitive Science, University of Edinburgh, 1996.
- [8] M. J. L. Orr, "Optimising the widths of radial basis functions," *Proceedings of the Fifth Brazilian Symposium on Neural Networks*, 1998.
- [9] M. J. L. Orr, "Recent advances in radial basis function networks," Technical report, Center for Cognitive Science, University of Edinburgh, 1999.
- [10] M. J. L. Orr, "Matlab functions for radial basis function networks," <http://www.anc.ed.ac.uk/~mjo/software/rbf.zip>.
- [11] T. Poggio and F. Girosi, "Networks for approximation and learning," *IEEE Proceedings*, vol. 78, no. 9, pp. 1481-1497, 1990.
- [12] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.