The Fast Neural Network Solution for Problems Based on Slow Genetic Algorithm Solutions

Yan Wang and Yilong Lu
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore 639798
eMail: eylu@ntu.edu.sg

Abstract - This paper presents a study of using Radial Basis Function Neural Network, that is trained by a finite number of off-line slow genetic algorithm solutions, for infinite number of fast approximate solutions. This approach makes powerful yet slow genetic algorithm solutions possible for real time problems.

I. Introduction

Genetic Algorithms (GAs) are stochastic search algorithms that mimic the process of natural selection and genetics. They were originally developed by Holland [1] to study the adaptive process of natural systems and to design software systems with adaptive behavior. In the simplest form, a genetic algorithm requires a string representation to code the parameter space, a fitness function to evaluate the strings, a set of genetic operators to create new strings, and a set of probabilities to control the genetic operators [2].

Since early 1990s, GAs have received ever increasing interests as effective tools for complicated problems that are difficult or impossible to be solved by other methods. GAs have been used in a very wide range of areas. However, one of the major drawbacks of genetic algorithms is the slow convergence, which prevents GAs to be used in real time application system.

In our previous studies [3]-[5], we have successfully applied GAs to solve some challenging array beamforming problems such as side lobe suppression, array element failure correction, low sidelobe wide null steering, and etc. As GA solutions are normally very slow and are traditionally considered not practical for real-time application.

In order to make GA solutions possible for real time problems, we propose, in this study, to use a suitable neural network (NN) as the nonlinear approximate function of a complicated problem. The neural network could be trained by a finite number of off-line slow GA solutions and the trained neural work could be used for infinite number of real time solutions of the problem that could only be solved by genetic algorithms.

II. Neural Networks

Neural Networks have a very wide range of applications in a diverged areas. Many of the applications of neural networks, particularly in the areas of nonlinear system identification and control, reduce to the problem of approximating unknown functions of one or more variables from discrete measurements [8]-[11]. A number of studies have established that multilayer feedforward neural networks, with a variety of activation functions, serve as universal approximators. Continuous nonlinear functionals can be approximated with feedforward neural networks, and these can be used to directly approximate the outputs of dynamical systems [11]. Radial Basis Function Neural Network (RBFNN) is easier to initialize and to train than multilayer perceptrons [6],[7]. This study tries to a RBFNN for possible on-line real time solutions of problems that have to be solved by off-line slow genetic algorithms.

RBFNN is a kind of special feedforward Backpropagation Neural Network (BPNN), and it may require more neurons than standard BPNN. It works best when many training vectors are available. A BPNN operates in a two-step sequence during training [12],[13]. First, an input pattern is presented to the network’s input layer. The resulting activity flows through the network from layer to layer until the network’s response is generated at the output layer. In the second step, the network’s output is compared to the desired output for that particular in-
put pattern. If it is not correct, an error is generated, which is passed or propagated backward through the network from the output layer back to the input layer, with the weights on the intralayer connections being modified as the error backpropagates, as shown in Fig. 1. A BPNN, like any other neural network, is determined by the connections between the neurons, the transfer function used by the neurons, and the weight change law that controls training of the network.

Typically, a BPNN consists of three or more fully connected layers. Every neuron in each layer has an output connection to every neuron in the next layer. Generally, there are no connections between the neurons within a single layer, and usually the neurons connect only to neurons in the immediately following layer. Thus, the input-layer neurons normally connect only to the middle-layer neurons, which in turn connect only to the output-layer neurons.

Each neuron determines its output in the following fashion. First, it computes the weighted input of its net. Next, this input value is passed through the activation function, sometimes called a “squashing” function. For a RBFNN, the transfer function is:

$$f(n) = e^{-n^2}$$  \hfill (1)

The only remaining element to define in the network is the weight change law, or learning law. The generalized delta rule specifies the change in a given connection weight as

$$\Delta w_{ij} = \beta E f(n)$$  \hfill (2)

where $E$ is the error for this neuron, $\beta$ is the learning constant (a parameter between zero and one) and $f(n)$ is the input to the neuron. The error in the output layer is passed back to the middle-layer neurons and is weighted by the same connection weights that modified the forward activation flow. The net error in each middle-layer neuron is thus the weighted sum of the error contributions from each of the output-layer neurons.

Fig. 2 is the two layers architecture of a RBFNN. It consists of two layers: a hidden radial basis layer and an output linear layer.

The function approximation can be illustrated clearly by a very simple example shown in Fig. 3. In this example, a set of 21 numbers $x = [1:0.5:10]$ and its corresponding cosine function data $y = \cos(x)$ are used as the training data for a RBFNN. The objective is to train the RBFNN by these data to approximate the cosine function. $x$ and $y$ are used as training input and training output, respectively. The well trained RBFNN should be a good approximate function as cosine. For example, for the trained network, given an arbitrary input, such as $x_1 = 2.2$, the network can output an value $y_1 = -0.5627$ that is compared to the exact solution $y = \cos(2.2) = -0.5885$. The error is about 4.6%. From Fig. 3, we can see that RBFNN have achieved a good continuous nonlinear function approximation.

In our experiment of low sidelobe wide null steering problems [5], the input of RBFNN is a template of $N$ variables and output of RBFNN is a weighting vector of $M$ dimensions, corresponding to the number of array sensors. This is a complicated problem which has so far only be solved by genetic algorithms. Through the experiment, RBFNN is proved effective to resolve such $N\cdot M$ dimensional approximation problem. After the RBFNN is trained by a number of GA solutions, it can work as a fast optimization tool that can generate the proper weights in the $M$ dimensional space for other arbitrary wide null steering without further time consuming GA solutions.
Fig. 3. simple example of RBFNN function approximation

III. GA-RBFNN Based Low Sidelobe Wide Null Steering

A. Beamforming Formulation

For an arbitrary array of \( M \) element, the pattern function of the array factor can be expressed by the general function,

\[
F(\theta, \phi) = w^T S(\theta, \phi)
\]

where \( w \) is a column vector with \( M \) complex weighting coefficients, \( S \) is the generic steering column vector, \( r_m \) is the element location vectors, \( \hat{a}_r \) is unit vector of distance ray of the spherical coordinate, and \( \theta \) and \( \phi \) are the elevation and azimuth angles, respectively.

The objective of adaptive digital beamforming is to find proper weighting coefficients \( w \) to achieve desired pattern shape, including beam pointing direction, beamwidth, sidelobe level, null pointing direction, null depth, and etc. Fig. 4 shows an example of wide nulling.

B. Getting Weights from GA as Training Data for NN

GA has been proposed and demonstrated to be effective for the calculation of weights for wide null steering [5]. However, GA solutions are usually very slow and not suitable for real-time application. In this paper, we propose to use off-line slow GA solutions as the training data for neural network and the trained neural network can be used for on-line fast applications.

In this approach, a series of weighting vectors (chromosomes) for different wide null steering are solved by the GA that uses direct real-number coding and floating-point genetic operations.

In the GA computation, an initial population of 100 random chromosomes is generated. A weighting vector of a Taylor array close to the original pattern is added to replace the weakest individual among the initial population. The insertion helps to improve the rate of convergence.

The Emperor-Selective (EMS) mating scheme [4],[5] is used in experiment. Extrapolation Crossover and Non-uniform Up-slope Mutation are used for crossover and mutation, respectively.

A template (formed by null pointing direction, null width, null depth, sidelobe level, major beam pointing direction, beam width, as illustrated in Fig. 4) is cast over the array pattern produced by each candidate to compute their cumulative difference as a form of fitness measure [5]. The computing time for GA solutions depends on the desired fitness level or the maximum number of generations. Typically, the solution for a satisfactory weighting vector takes about 1 to 2 hours on Pentium 700 MHz PC under Matlab environment [14].

C. RBFNN Beamformer

This section briefly describes how to apply the GA-RBFNN approach to the low sidelobe wide null steering beamforming problem. Traditionally, adaptive nulling is within a fairly narrow angle to reject a strong interfering source, at a specific az-
imath direction. But in most applications, interfering signals could come from more than one direction. Due to increasing electromagnetic pollution of the environment, beamforming techniques that allow the placement of more than one null or a wide null in the antenna pattern at specific jamming directions are becoming more important.

In this example, the neural network is used as a tool to generate in real time the array weights (w). This neural network should have been well trained in advance by off-line GA solutions for wide null steering [5]. When a desired antenna beam pattern (F_N) is present at the input layer of trained RBFNN, the array weighting vector w is produced at the output layer. The RBFNN is designed to perform an input-output mapping trained with examples \{F^{(i)} \Rightarrow w^{(i)} : i = 1, 2, ..., N_t \}, where N_t stands for the number of examples contained in the training set. The trained RBFNN is supposed to generalize outputs when given inputs beam patterns are not among the GA solutions.

D. Training the Neural Network

The data for training the RBFNN is a set of finite number of GA solutions. First, given desired pattern matrix \{F^{(i)}; i = 1, 2, ..., N_t \}, the array weighting vectors \{w^{(i)}; i = 1, 2, ..., N_t \} are calculated using GA as the required training input/output pairs of the training set. The desired antenna pattern can be denoted by such input parameters as: null pointing direction, null width, null depth, sidelobe level, major beam pointing direction, and beam width.

After training, the resultant RBFNN should have stored the characters and relationships among these data. Because of the approximation function of the neural network, the original low sidelobe wide null steering problem can be approximately represented by the trained RBFNN. The successfully trained NN should have the follow character: when given the NN any other desired antenna pattern that is not among the training data, the NN should output the array weights that is able to produce the desired pattern or in other words they are good approximations of the array weights from GA solution of the same pattern.

Properly-trained neural network tends to give reasonable answers when presented with inputs that they have never seen. This generalization property makes it possible to train a network on a representative set of input/target pairs and get good results without training the network on all possible input/output pairs.

The desired benefit is that the trained NN can generate infinite number of fast solutions that can be implemented for on-line real time application.

E. Numerical Experiment

In simulation, a 32-element half-wavelength spaced linear array is used as the working example. In this example, two different null widths (20 and 30 degrees) are considered. To train the RBFNN, 108-sets of GA solutions are obtained first as input/output training data for the RBFNN. As shown in Table I, the 108 cases can be presented in 12 groups. In each group, 9 discrete null pointing directions are chosen, i.e., the left edges of nulls (θ₁) are from 10 degree to 50 degree at a step of 5 degrees.

<table>
<thead>
<tr>
<th>Null width (degree)</th>
<th>Null depth (dB)</th>
<th>Sidelobe level (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td>-20</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>-25</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>-30</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>-20</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>-30</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>-20</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>-25</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>-30</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>-20</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>-25</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>-30</td>
</tr>
</tbody>
</table>

Note: Each with 9 θ₁ (from 10 to 50 with a step of 5).

Fig. 5 and Fig. 6 show some of the test results of the low sidelobe wide null steering to illustrate the effectiveness of the above trained RBFNN. The rectangular lines in the Figures show the templates of the desired patterns.

In Fig. 5, the input parameter vectors to the RBFNN is denoted by the input parameter vectors [null pointing direction, null width, null depth, sidelobe level, major beam pointing direction, beam width] = [47, 20, 20, -30, 90, 7], [48, 20, 20, -30, 90, 7], [49, 20, 20, -30, 90, 7], respectively.

In Fig. 6, the input parameter vectors are [38, 30, 20, -30, 90, 9], [39, 30, 20, -30, 90, 9], [41, 30, 20, -30, 90, 9], respectively.

Be noticed that those input patterns are not among the training examples. We have also conducted exhaust test for θ₁ from 10 degrees to 50 degrees at a step of 1 degree and the test results are very good. The time cost using trained RBFNN to compute each weighting vector is less than 0.02 second under Matlab environment on a Pentium 700 MHz PC. Please also note that in this experiment, the weighting coefficients are confined to real numbers, so the nulls look symmetrical about the major beam. We will test the complex weights and
Fig. 5. Examples of low sidelobe wide null steering with null width of 20 degrees.

Fig. 6. Examples of low sidelobe wide null steering with null width of 30 degrees.
better results are expected.

For a comparison, we consider an example with beam pattern parameters \([\text{null pointing direction, null width, null depth, sidelobe level, major beam pointing direction, beam width}] = [47, 20, 20, –30, 90, 7]\). The pattern of the RBFNN array together with the patterns of the Chebyshev array and Taylor array for the same desired sidelobe level are shown in Fig. 7. The 3-dB beamwidth of the Chebyshev array, the Taylor array, and GA-RBFNN array are 3.88°, 4.18°, and 4.15°, respectively. The 3-dB beamwidth of the RBFNN array is slightly wider than that of the Chebyshev array and narrower than that of the Taylor array however the RBFNN array can provide a wide deep null in the desired range of directions without increase the sidelobe level. This feature is useful for adaptive beamforming to suppress strong interferences from varied wide angles.

IV. Conclusion

An approach of using neural network to make powerful yet slow genetic algorithm solutions possible for real time application is proposed. In this approach the Radial Basis Function Neural Network is selected to be trained first by a finite number of representative off-line slow genetic algorithm solutions, the trained neural network is then used as a nonlinear approximate function for an infinite number of fast solutions of the original problem without using genetic algorithm solver.

The proposed approach is applied to low sidelobe wide null steering in array beamforming and numerical experiments show that the GA-RBFNN approach is effective and it is possible to solve problems, in real time, that were not possible before.

References