INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 8, Number 3, Pages 484–495

CONVERGENCE OF GRADIENT METHOD FOR DOUBLE PARALLEL FEEDFORWARD NEURAL NETWORK

JIAN WANG, WEI WU, ZHENGXUE LI, AND LONG LI

Abstract. The deterministic convergence for a Double Parallel Feedforward Neural Network (DPFNN) is studied. DPFNN is a parallel connection of a multi-layer feedforward neural network and a single layer feedforward neural network. Gradient method is used for training DPFNN with finite training sample set. The monotonicity of the error function in the training iteration is proved. Then, some weak and strong convergence results are obtained, indicating that the gradient of the error function tends to zero and the weight sequence goes to a fixed point, respectively. Numerical examples are provided, which support our theoretical findings and demonstrate that DPFNN has faster convergence speed and better generalization capability than the common feedforward neural network.

Key Words. Double parallel feedforward neural network, gradient method, monotonicity, convergence.

1. Introduction

A Double Parallel Feedforward Neural Network (DPFNN) is a parallel connection of a multi-layer feedforward neural network and a single layer feedforward neural network. In a DPFNN, the output nodes not only receive the recodification of the external information through the hidden nodes, but also receive the external information itself directly through the input nodes. DPFNN involves a paratactic relationship between linear and nonlinear mappings [4, 1]. As in the case for the common feedforward neural networks [18, 13, 19, 20], the most widely used learning method for DPFNN remains to be the gradient method [17, 10, 15, 2]. It is shown (cf. [5]) that the training speed and accuracy are greatly improved for DPFNN compared with corresponding multi-layer feedforward neural networks [8, 12, 11, 3, 9, 7]. A double parallel feedforward process neural network with similar structure and updating rule as DPFNN is proposed in [22]. In [16], an alternate learning iterative algorithm for DPFNN is presented. The truncation error caused by word length on the accuracy of DPFNN is analyzed in [6].

We are concerned in this paper with the convergence of the gradient method for training DPFNN. In particular, we first prove the monotonicity of the error function in the gradient learning iteration for DPFNN. Then, some weak and strong convergence results are obtained, indicating that the gradient of the error function tends to zero and the weight sequence goes to a fixed point, respectively. Some supporting numerical examples are also provided, which support our theoretical

Received by the editors May 4, 2009 and, in revised form, March 22, 2011.
2000 *Mathematics Subject Classification.* 68W40, 92B20, 62M45.
This research was supported by the National Natural Science Foundation of China (No.10871220).

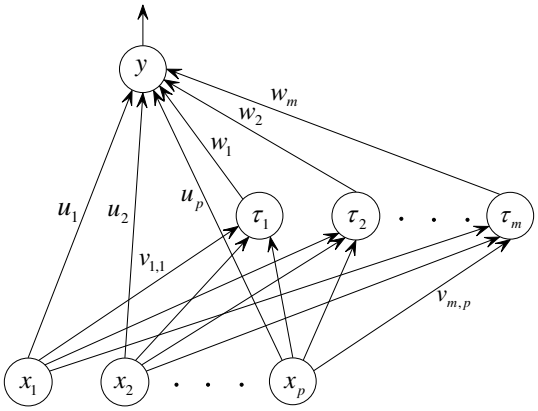

FIGURE 1. Topological Structure of DPFNN.

findings and demonstrate that DPFNN has faster convergence speed and better generalization capability than the common feedforward neural network.

The rest part of this paper is organized as follows. The structure of and the gradient method for DPFNN are introduced in Section 1. In Section 2 the convergence results are presented. Section 3 provides a few numerical examples to support our theoretical findings. Some brief conclusions are drawn in Section 4. Finally, an appendix is given, in which the details of the proof are gathered.

2. Double Parallel Feedforward Neural Networks

Figure 1 shows the DPFNN structure considered in this paper. It is a three-layer network with p input nodes, m hidden nodes and 1 output node. We denote the weight vector connecting the hidden layer and the output layer by $\mathbf{w} = (w_1, \cdots, w_m)^T \in \mathbb{R}^m$, and the weight matrix connecting the input layer and the hidden layer by $\mathbf{V} = (v_{i,j})_{m\times p}$, where $\mathbf{v}_i = (v_{i,1}, \cdots, v_{i,p})^T \in \mathbb{R}^p$ is the weight vector connecting the input layer and the i-th node of the hidden layer. Similarly, we denote the weight vector connecting the input layer and the output layer by $\mathbf{u} = (u_1, \cdots, u_p)^T \in \mathbb{R}^p$.

For simplicity, all the weight vectors are incorporated into a total weight vector $\mathbf{W} = (\mathbf{u}^T, \mathbf{v}_1^T, \cdots, \mathbf{v}_m^T, \mathbf{w}^T)^T \in \mathbb{R}^{p+mp+m}$. Let $g : R \to R$ be an activation function for the hidden and the output layers. For any $\mathbf{z} = (z_1, \cdots, z_m)^T \in \mathbb{R}^m$, we define

$$G(\mathbf{z}) = (g(z_1), g(z_2), \cdots, g(z_m))^T \in \mathbb{R}^m. \tag{1}$$

For any given input vector $\mathbf{x} \in \mathbb{R}^p$, the actual output $y \in \mathbb{R}$ of the neural system is computed by

$$y = g(\mathbf{w} \cdot G(\mathbf{V}\mathbf{x}) + \mathbf{u} \cdot \mathbf{x}). \tag{2}$$

We remark that the bias terms should be involved in the neural system. However, following a common strategy, we set the last component of, say, the input vector $\mathbf{x}$ to be -1, and so the last component of $\mathbf{v}_i$ corresponds to the bias term. This strategy allows us not to write explicitly the bias terms in the description of our problem.