Appendix A

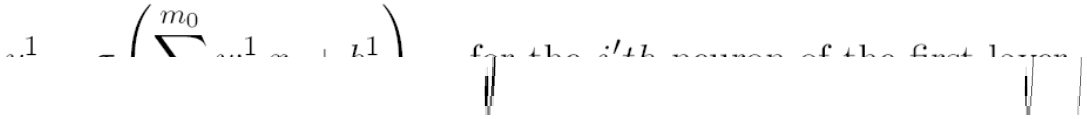
Data Flow Design for the Backpropagation Algorithm

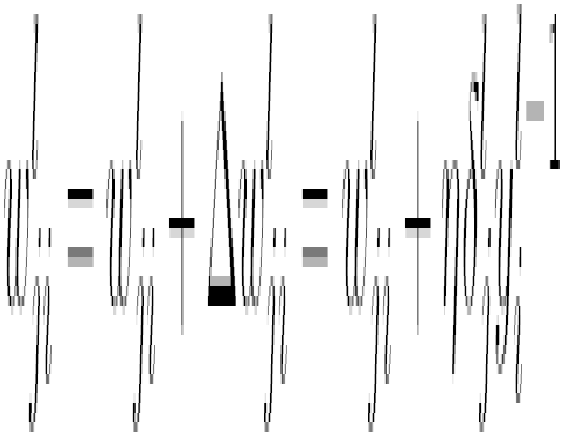
Abstract - We report a data flow design for the multilayer network. Both back-propagation(BP) learning and feedforward computing are constructed with a single basic module where each neuron is regarded as a module. This design can be extended to various networks.

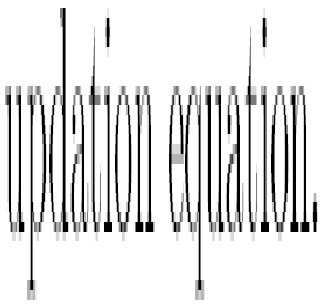
1. Introduction

Many data flow designs for neural networks have been developed for various purposes with varied successes, such as that for the committee machine. We report our design in this paper and omit a full review of them. A back-propagation algorithm trains network layer by layer doing forward and backward computations. According to the algorithm the updation formulas are:

forward computation: (1)







In the above equations w denotes the weight between two neurons. d is the desired response. *x* is the input. *y* denotes the neuron's output. σ is the active function. η is a tunable learning rate. *l* denotes the number of layer, where 1 denotes the first hidden layer and L is the output layer. *i* or *j* denote the number of neuron in each layer. So, y_{lj} is the output of the *j*'th neuron in the *l*'th hidden layer, w_{ji}^{l} is the weight between the *j*'th neuron in the *l*'th layer and the *i*'th neuron in the *(l-1)*'th layer. b_j^l is the *j*'th neuron's bias. d_j^l is the desired response of the *j*'th neuron in the *l*'th layer. δ_j^l is the *j*'th neuron's delta value for weight correction in the *l*'th layer. m_0 is the number of neurons in input layer, m_{l-1} is the number of neurons in the *(l-1)*'th layer . All neurons use these equations to improve their weights. Each neuron use the outputs of all neurons in the next precedent layer as inputs. We will isolate each neuron with all its weights, inputs, desired response, and output. This allows us to implement the BP algorithm on distribute parallel machine. In the next section, we present the basic module for a single neuron. Then we show a data flow structure for multilayer networks constructed with such basic module.

2. The basic module

Fig. 1 shows the diagram of a single neuron. The forward equation is:

$$y^t = \sigma\left(\sum_{i=0}^{N} w_i^t x_i^t + b\right) \tag{3}$$

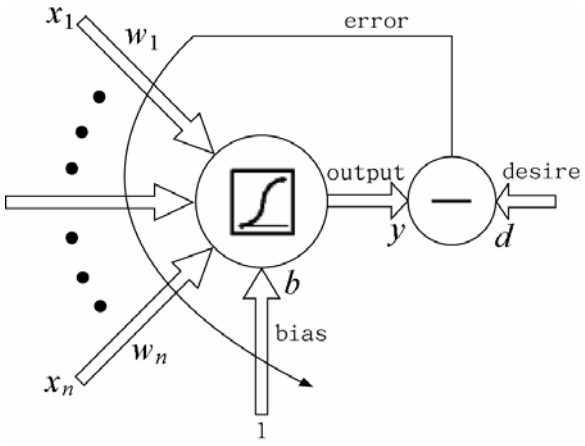


Fig. 1. Single neuron diagram.

where t denotes time and the active function is temporarily set to the bipolar sigmoid function $\sigma(u)=\frac{2}{1+e^{-u}}-1$. According to the delta learning rule (2)(3), the error-correction function is defined by:

$$\Delta w^t = \eta \frac{1}{2}\left(d^t - y^t\right)\left(1-(y^t)^2\right)x^t \tag{4}$$

And the weight is updated according to:

$$w^{t+1} = w^t + \Delta w^t \tag{5}$$

Combining Eq. (3)(4) and setting F be the updation function for this neuron, we obtain:

$$(\Delta w^t, y^t) = F(w^t, x^t, d^t) \tag{6}$$

where F uses w(weight), x(input), and d(desire response) as its inputs and Δw^t and y^t as it's outputs. The data structure for this neuron is plotted in Fig. 2. This structure is well known among the engineering society.

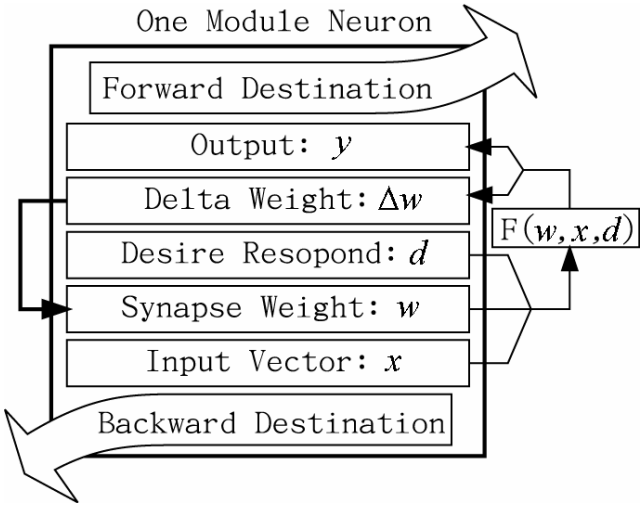


Fig. 2. One module neuron data structure.

3. The modular design for artificial neural networks

We can use this basic module to construct various kinds of neural networks, such as multiplayer network, multilayer network with jump connection, recurrent network and selforganized map. In this section we show how to construct a multilayer network and recurrent network.

3.1Multilayer network

A multilayer network can be transformed into its modular form in O(n) time, where n is the number of total neurons. With a pointer supporting language, such as C, we can allocate a memory space for each neuron and maintain a pointer pointing to it. The algorithm is:

algorithm Modular Transform
for L=1 to number of layers
for N=1 to the number of neurons in layer L
Neu ← allocate a memory space for each neuron.
Store all data of this neuron(L,N) in Neu.
Set Neu→forward destination point to neurons in next layer.
Set Neu→backward destination to neurons in precedent layer.
Look up table(N,L)← Neu's location.
end.
end.

It is a little cost of memory doing this modular transform. Usually the BP algorithm is implemented using a matrix (or an array) to store weights, input vector and output vector. Each entry in the matrix corresponds to a neuron's relative position in the network. Instead of using matrix, we maintain a pointer for each neuron which contains the synapses to all linked neurons. The key part of the module is that the desired response for each neuron must be given in advance, not only for the neuron in the output layer. Therefore, the BP algorithm (equation 1 and 2) must be reformed in such a way that every hidden neuron can be treated like an independent neuron as long as we can calculate its desired response. For this response, observe the two equations:

$$\delta_j^L = \frac{1}{2}\,(d_j^L - y_j^L)\left(1 - (y_j^L)^2\right), \tag{7}$$

$$\delta_j^l = \frac{1}{2}\,\left(1 - (y_j^l)^2\right)\sum_{i=1}^{m_{l+1}} \delta_i^{l+1} w_{ij}^{l+1}. \tag{8}$$

Equation 10 and 11 are the backward delta equation in the BP algorithm using the bipolar sigmoid function. Symbols are defined same as preceding section. Equation (7) is for the neurons in the output layer and equation (8) is for hidden layer. Simplify these two equations, where we regard y_j^L and y_j^l as the same role, we obtain the

desired response for each neuron in the hidden layer,

$$d_j^l = y_j^l + \sum_{i=1}^{m_{l+1}} \delta_i^{l+1} w_{ij}^{l+1}. \tag{9}$$

According to $\Delta w_{ij}^L = \eta \delta_i^L y_i^{L-1}$, the delta rule for the neuron *i* in the output layer is:

$$\delta_i^L = \frac{\Delta w_{ij}^L}{\eta y_i^{L\text{-}1}} \tag{10}$$

Substitute equation (10) into (9) the desired response for the *j*'th hidden neuron is:

$$d_j^l = y_j^l + \sum_{i=1}^{m_{l+1}} \frac{\Delta w_{ij}^{l+1}}{\eta y_i^l} w_{ij}^{l+1}, \qquad \{l+1 = 2....L\}. \tag{11}$$

With equation 14 we obtain all neurons' desired responses no matter what layer it belonging to. Therefore each neuron can be treated separately. To our knowledge this equation has not been discussed before.

In figure 4, we illustrate a flow chart of the modular design for the BP algorithm. The main difference between the formal BP algorithm is that we calculate each neuron's desired response before adjust its weights. Figure 5 shows an example of a 1-3-2 modular design network. Similar to the multilayer feed forward network, a multilayer network can have jump connection (see figure 3) from lower level to higher level. It's forward and backward equations are similar to the BP equations 1 and 2. Its modular design is similar to that for the multiplayer feed forward network.

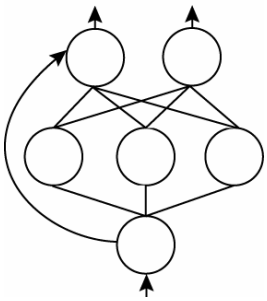


Fig. 3. An example of a multilayer network with jump connection

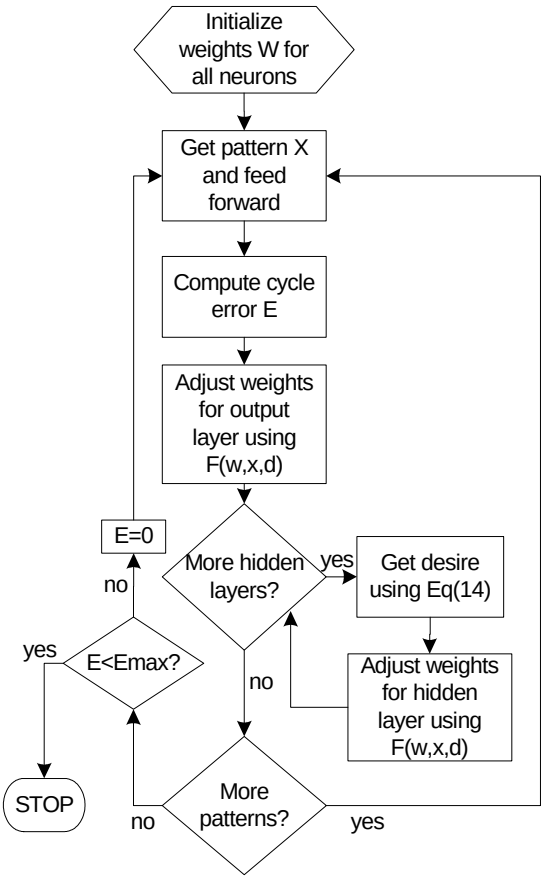


Fig. 4. Modified BP training flow of modular design network.

3.2 Recurrent network

A recurrent network uses its outputs as its inputs. We show the modular design for training a recurrent network in Figure 6. The training procedure starts by feeding an input vector x into the modular network. The desired response is set to a target sequence. The output of the network is feedback to itself as the next input in each iteration. The procedure will be continued until the error reduced to a satisfiable range. We treat recurrent network as a normal feed forward network, where we connect it's output destination back to itself.

The modular design is particularly useful for the data flow machine. It is believed to achieve high degree of parallel computation. The main achievement is that we can decompose neural networks into small modules which enable us to feed each module into multi-speed processors that can conform the spirit of data flow machine. The self-organization neural network can cope with the data flow machine structure with less modifications and we omit its discussion.

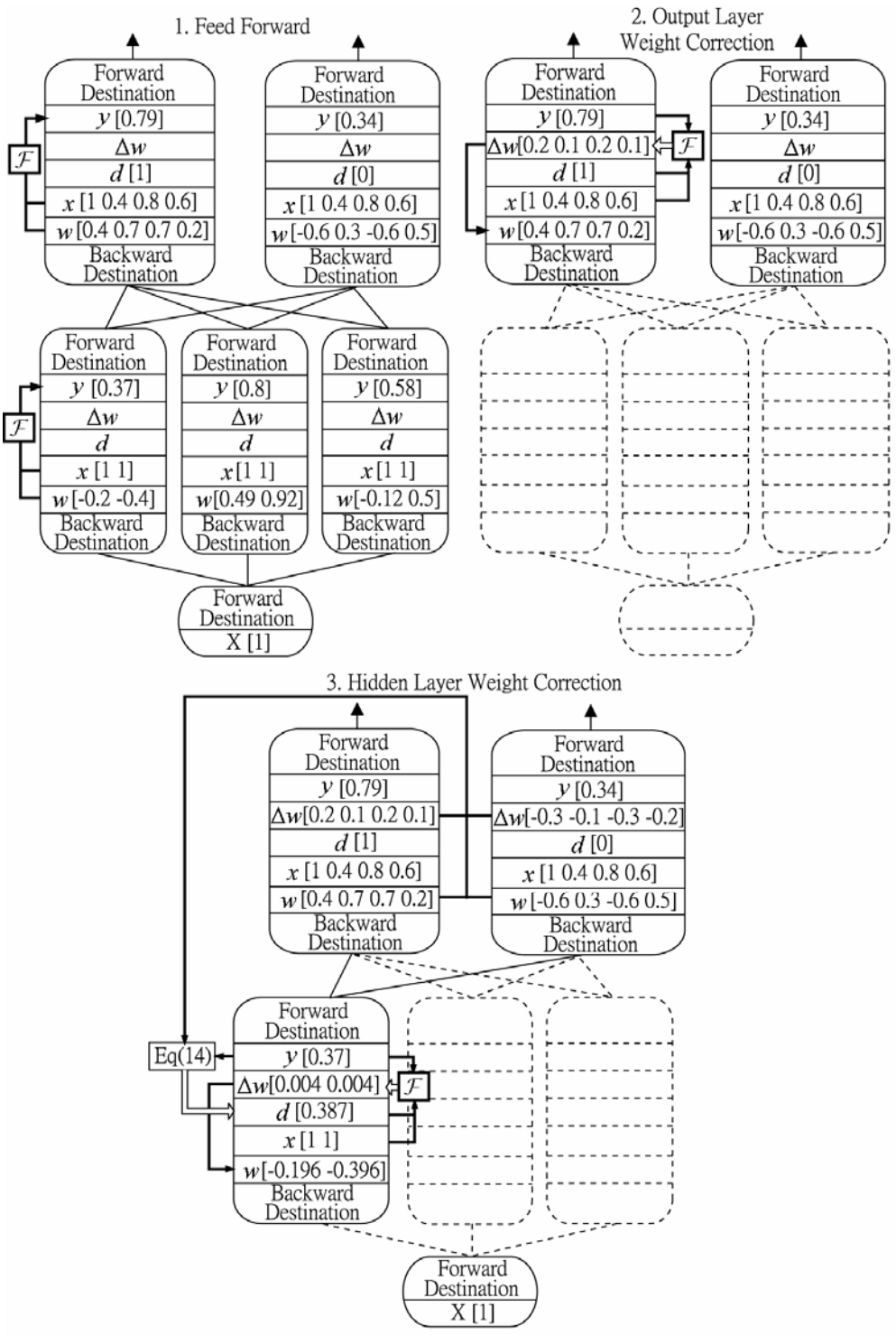


Fig. 5. Training procedure of modular design network. The values in step 1 are randomly generated. Succeeding steps change its values according to initial step.

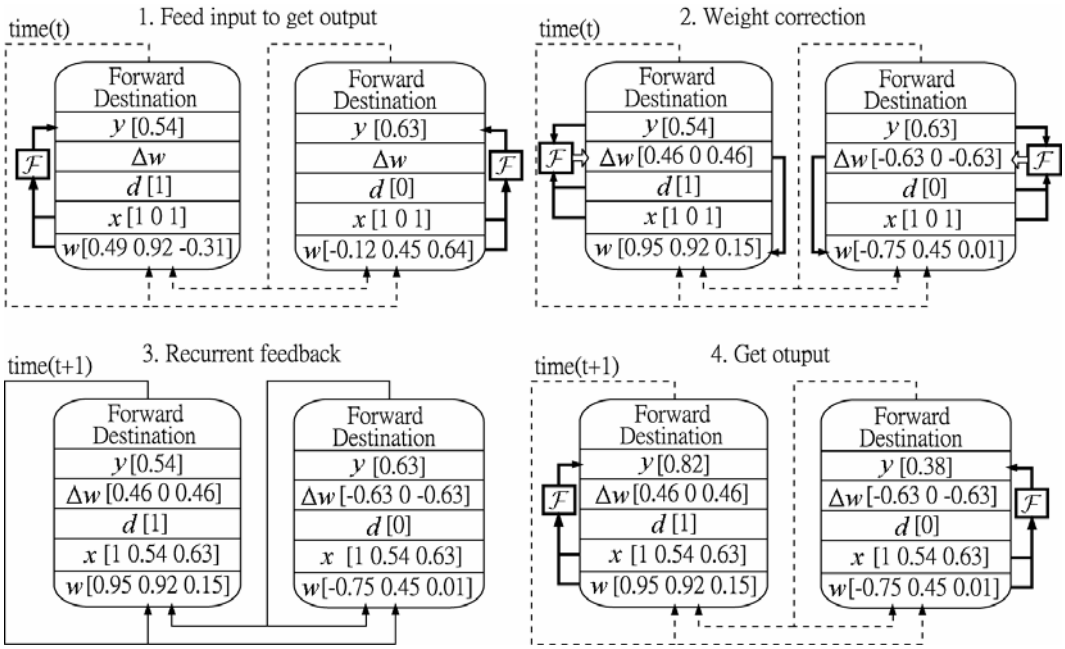


Fig. 6. The training procedure of recurrent network using modular design. The values in step 1 are randomly generated. Succeeding steps change its values according to initial step.