Implementation of Dynamics Neural Network in Solving Inverse Function

C. K. Ang*, M. I. Solihin and T. H. Tan
Faculty of Engineering, Technology and Built Environment, UCSI University, Kuala Lumpur, Malaysia; ack_kit@hotmail.com, mahmudis@ucsiuniversity.edu.my, tanth@ucsiuniversity.edu.my

Abstract

Back-Propagation Neural Network (BPNN) has been widely used in solving nonlinear problems. However, there are some limitations in using conventional BPNN especially for high order nonlinear problems. Dynamic Back-Propagation Neural Network (DBPNN) is proposed in this paper to improve the performance of conventional BPNN. Its adaptive learning ability is closer to human being learning behavior in comparing to conventional BPNN. Few simulations have been run to test the robustness of DBPNN and the results are compared to the conventional BPNN.

Keywords: Artificial Intelligences, Neural Networks, Nonlinear Function

1. Introduction

Back-Propagation Neural Network (BPNN) is one of the machine learning approaches which involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy. The learning capabilities of BPNN enable the system to improve the performance over time and BPNN is easy to be implemented in comparing to other network models such as recurrent network, Bidirectional Associative Memory (BAM), etc.

Basically, there are four steps to implement the BPNN which are initialization, activation, weight training and iteration. The most challenging step in applying BPNN is the training process. Conventionally, it is compulsory to have sufficient training data for training a BPNN in order to get an accurate result and the number of training data will range from 300 to 400 samples. One of the limitations of training BPNN is the training input is only allowed to map with one training output which is considered as one to one mapping module. In other words, conventional BPNN is not suitable for the system which consists of multiple outputs for the same input. Besides, there is no sensitivity consideration for conventional BPNN. For an example, human is sensitive and responsive to latest events and insensitive to earlier or past events and conventional BPNN does not possesses this instinct.

As compared to conventional BPNN, DBPNN which is introduced in this paper is more suitable for the system which is not in one to one mapping module and DBPNN is sensitive and responsive to latest events.

2. Neural Networks Model

2.1 Conventional Back-propagation Neural Network Model

As mentioned earlier, there are four steps to develop the BPNN which are initialization, activation, weight training and iteration. The topology of conventional BPNN is shown in Figure 1.

Initialization is the procedure to set the weights and threshold levels of the network randomly. The BPNN can be activated by presenting a training set data with inputs \( \{x_1(p), x_2(p), \ldots, x_n(p)\} \) and desired outputs \( \{y_1(p), y_2(p), \ldots, y_l(p)\} \).
It is assumed that the BPNN consists of three layers which are input layer, hidden layer, and output layer. The actual outputs of the neurons in the hidden layer will be

$$y_j(p) = \text{sigmoid} \left[ \sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j \right]$$

(1)

where $n$ is the number of inputs for neuron $j$ in the hidden layer, and $\text{sigmoid}$ is the sigmoid activation function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

(2)

where

$$X = \sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j$$

(3)

For the output layers, the actual outputs of the neurons is given by

$$y_k(p) = \text{sigmoid} \left[ \sum_{j=1}^{m} x_{jk}(p) \times w_{jk}(p) - \theta_k \right]$$

(4)

where $m$ is the number of inputs for neuron $k$ in the output layer. An error signal is represented by

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

(5)

will be generated from neuron $k$ at output layer and propagated backward for updating the weights. For updating the weights at the output neurons with learning rate $\alpha$, the error gradient for the neurons in the output layer will be

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p)$$

(6)

The weight correction will be

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$$

(7)

Thus, the updated weights at the output neurons will be

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

(8)

Similar to output layer, for updating the weights at the hidden neurons with learning rate $\alpha$, the error gradient for the neurons in hidden layer will be

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{m} \delta_k(p) \times w_{jk}(p)$$

(9)

The weight correction will be

$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p)$$

(10)

and thus, the updated weights at the hidden neurons can be written as

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

(11)

The same procedures can be repeated by increasing the iteration $p$ until the output error criterion is satisfied. Once the error criterion is satisfied, another new set of inputs and desired outputs will be trained by the same procedures until all training samples reach the minimum error criterion. Assuming that there are $s$ samples in total, BPNN expects to let all those $s$ samples to reach the minimum error and there is no special priority for any of the samples.

**Figure 1.** Conventional back-propagation neural network topology.

### 2.2 Proposed Dynamic Back-propagation Neural Network Model

The training algorithm for DBPNN is quite similar to conventional BPNN model. However, DBPNN does not train all the training samples in one time. In order to make DBPNN sensitive to current events, DBPNN trains the samples in sequences by switching the training samples window.

Assuming that there are $s$ training samples in total, the training inputs for each sample will be
The same as BPNN, the first stage for DBPNN will be the initialization; the weight and bias for hidden and output neurons are set randomly for \(N_1\) at first iteration. Once the training reach minimum error criterion, the weight and bias for hidden and output neurons will be stored in matrices, \(w_{\text{hidden}}\), \(w_{\text{output}}\), \(b_{\text{hidden}}\), and \(b_{\text{output}}\). These weight and bias matrices will be used as an initial weight and bias values for \(N_2\) training samples. The same procedures are repeated until the samples in group \(N_{s-q+1}\) are trained completely. The topology of DBPNN is shown in Figure 2. It is very similar to conventional BPNN but it trains the samples in small group by switching the training samples in sequence. This creates a sensitivity instinct for neural network in predicting latest event.

3. Results and Discussion

The comparison in term of performance is made between DBPNN and conventional BPNN to show the robustness of DBPNN. Assuming that there is a dynamics nonlinear function, \(f(x)\). It is easy to get the value for function \(f(x)\) by providing the value of \(x\). In fact, there will be more chal-

### Table 1. Condition 1

| Condition: | Domain for \(x\): \(\{x \mid 0 \leq x \leq 10\}\) |
|------------|--------------------------------------------------|
| Samples were collected evenly from different interval |

| \(f(x)\) Value | Desired \(x\) | \(x\) BPNN | \(y\) BPNN | Error BPNN | Error DBPNN |
|----------------|---------------|-------------|-------------|-------------|-------------|
| 3.1661         | 1.3           | 1.4067      | 1.1095      | 0.1067      | 0.2972      |
| 7.4621         | 1.7           | 1.6097      | 1.8608      | 0.0903      | 0.2511      |
| 40.9376        | 2.6           | 2.5907      | 2.4384      | 0.0093      | 0.1523      |
| 64.3181        | 2.9           | 2.9059      | 2.7647      | 0.0059      | 0.1412      |
| 124.0736       | 3.4           | 3.3971      | 3.4348      | 0.0029      | 0.0377      |
| 295.5296       | 4.2           | 4.1995      | 4.4064      | 0.0005      | 0.2069      |
| 554.4701       | 4.9           | 4.9         | 4.969       | 0           | 0.069       |
| 954.0896       | 5.6           | 5.6002      | 5.5918      | 0.0002      | 0.0084      |
| 1537.6         | 6.3           | 6.2998      | 6.4277      | 0.0002      | 0.1279      |
| 1855.9         | 6.6           | 6.6005      | 6.7932      | 0.0005      | 0.1927      |
| 3109.8         | 7.5           | 7.4995      | 7.2275      | 0.0005      | 0.272       |
| 3642.7         | 7.8           | 7.7996      | 7.5148      | 0.0004      | 0.2848      |
| 4910.2         | 8.4           | 8.4009      | 8.4045      | 0.0009      | 0.0036      |
| 6197           | 8.9           | 8.8996      | 8.9338      | 0.0004      | 0.0342      |
| 8760.8         | 9.7           | 9.7         | 9.6478      | 0           | 0.0522      |
### Table 2. Condition 2

Condition:

Domain for x: \( \{ x \mid 0 \leq x \leq 10 \} \)

First 30 samples from interval 0 to 5, last 70 samples from interval 5 to 10

| f(x) Value | Desired x | BPNN | DBPNN | Error BPNN | Error DBPNN |
|------------|-----------|------|-------|------------|-------------|
| 3.1661     | 1.3       | 1.1559 | 0     | 0.1441     | 1.3         |
| 7.4621     | 1.7       | 1.7611 | 0     | 0.0611     | 1.7         |
| 40.9376    | 2.6       | 2.6022 | 0     | 0.0022     | 2.6         |
| 64.3181    | 2.9       | 2.8978 | 0     | 0.0022     | 2.9         |
| 124.0736   | 3.4       | 3.3998 | 0     | 0.0002     | 3.4         |
| 295.5296   | 4.2       | 4.2016 | 0     | 0.8164     |             |
| 554.4701   | 4.9       | 4.9001 | 5.2622 | 1E-04      | 0.3622      |
| 954.0896   | 5.6       | 5.5994 | 5.634 | 0.0006     | 0.034       |
| 1537.6     | 6.3       | 6.3007 | 6.1632 | 0.0007     | 0.1368      |
| 1855.9     | 6.6       | 6.6008 | 6.4414 | 0.0008     | 0.1586      |
| 3109.8     | 7.5       | 7.5008 | 7.4338 | 0.0008     | 0.0662      |
| 3642.7     | 7.8       | 7.7997 | 7.796 | 0.0003     | 0.004       |
| 4910.2     | 8.4       | 8.4006 | 8.5047 | 0.0006     | 0.1047      |
| 6197       | 8.9       | 8.8999 | 9.0205 | 1E-04      | 0.1205      |
| 8760.8     | 9.7       | 9.6994 | 9.6005 | 0.0006     | 0.0995      |

### Table 3. Condition 3

Condition:

Domain for x: \( \{| x | -10 \leq x \leq 10 \} \)

First 50 samples in the domain \( \{ x \mid 0 \leq x \leq 10 \} \)

Last 50 samples in the domain \( \{ x \mid -10 \leq x \leq 0 \} \)

| f(x) Value | Desired x | BPNN | DBPNN | Error BPNN | Error DBPNN |
|------------|-----------|------|-------|------------|-------------|
| 9902       | -10       | 9.9492 | -9.7368 | 19.9492    | 0.2632      |
| 6482       | -9        | -0.0035 | -9.0271 | 8.9965     | 0.0271      |
| 4034       | -8        | -9.7663 | -7.9874 | 1.7663     | 0.0126      |
| 1262       | -6        | -0.4295 | -6.1122 | 5.5705     | 0.1122      |
| 602        | -5        | -1.9163 | -5.4395 | 3.0837     | 0.4395      |
| 242        | -4        | -0.0493 | -3.9242 | 3.9507     | 0.0758      |
| 14         | -2        | -0.0476 | -1.8736 | 1.9524     | 0.1264      |
| 2          | 0         | -0.0476 | -0.7031 | 0.0476     | 0.7031      |
lenging if we are asked to find the value of $x$ by providing the value of function $f(x)$ such as solving the inverse kinematic issue in robotic, path planning for robot, and variety of application with inverse function. However, if BPNN is used to solve this issue, it might cause BPNN to lead to chaotic behavior.

![Proposed dynamic neural network topology.](image)

For an example, assuming that there was a nonlinear function,

$$f(x) = x^4 - x^2 + 2$$  \hspace{1cm} (12)

we were asked to find the desired value of $x$ by proving the function $f(x)$. Around 100 samples in total were collected for training this function and another 15 samples for validation purpose. The performance of BPNN and DBPNN were evaluated in 3 different conditions. BPNN and DBPNN were used to predict the value of $x$ by providing the value of the function. The results are shown in the Table 1, Table 2 and Table 3.

In Table 1, it is shown that BPNN performed better than DBPNN but the errors were acceptable for DBPNN. The variation between the desired output and DBPNN output is small and the absolute error is less than 0.3 units. In Table 2, conventional BPNN is still able to predict the correct output as compared to DBPNN. For DBPNN, it is sensitive only to the samples which are in the domain $\{x | 5 \leq x \leq 10\}$ because the last 70 training samples are all fall in this domain. In Table 3, the domain of training samples is changed from positive value to negative value. BPNN is unable to adapt the changes because there is no dedicated priority for old or new training samples and resulting BPNN failed to predict the output. For DBPNN, it is able to predict the output and adapt the changes of training samples. In the experiment, DBPNN shows that the output is always a negative value because it has successfully adapted the changes during the training process.

In terms of applications, a practical dynamics system does not guarantee that it can provide a very consistent output all the time. Sometimes, there might be some errors occur due to the malfunctioning of hardware part, control part, electronic components, and etc. Hence, it is better to create a new adaptive model which can adapt the changes of the system output from time to time and DBPNN can be considered as one of the solutions for this issue.

### 4. Conclusion

In conclusion, the conventional BPNN has a drawback of unable to adapt to the changes of outputs, lack of sensitivity and responsive to latest events. Therefore, DBPNN is proposed in this paper, and the results have demonstrated that DBPNN performs better than conventional BPNN in adapting the changes of outputs and its adaptive learning ability is closer to human being learning behavior.

### 5. References

1. Tang SH, Ang CK, Mashohor S, Arrifin MKAM. Predicting the motion of a robot manipulator with unknown trajectories based on artificial neural network. Int J Adv Robot Syst. 2014; 11(176):1–9.
2. Ang CK, Tang SH, Mashohor S, Arrifin MKAM. Solving continuous trajectory and forward kinematics simultaneously based on artificial neural network. Int J Comput Commun. 2014; 9(3):253–60.
3. Kumar N, Panwar V, Sukanam N, Sharma SP, Borm JH. Neural network-based nonlinear tracking control of kinematically redundant robot manipulators. Math Comput Model. 2011; 53:1889–901.
4. Shah J, Rattan SS, Nakra BC. Kinematic analysis of 2-DOF planar robot using artificial neural network. World Acad Sci Eng Technol. 2011; 81:282–5.
5. Park H, Lee S, Chu B, Hong D. Obstacle avoidance for robotic excavators using a recurrent neural network. IEEE Int Conf on Smart Manufacturing Application; 2008. p. 585–90.
6. Wang J. Obstacle avoidance for kinematically redundant manipulators based on recurrent neural networks. 1st IEEE Int Conf on Intelligent Robotics and Application; 2008 Oct. p. 10–3.

7. Kosko B. Adaptive bidirectional associative memories. Appl Optics. 1987; 26:4947–60.

8. Kim BH. An adaptive neural network learning-based solution for the inverse kinematics of humanoid fingers. Int J Adv Robot Syst. 2014; 11:1–9.

9. Capisani LM, Ferrara A. Trajectory planning and second-order sliding mode motion/interaction control for robot manipulators in unknown environments. IEEE T Ind Electron. 2012; 29:3189–98.