Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Local Cluster Neural Network On-chip Training

Liang Zhang, Joaquin Sitte
School of Software Engineering and Data Communications,
Queensland University of Technology, Australia
E-mail: l1.zhang@student.qut.edu.au, j.sitte@qut.edu.au

***Abstract*— The local cluster neural network is a feedforward RBF network that has been implemented in analogue neural net chip. The LCNN chip can be trained by chip-in-the-loop training and this training method has been demonstrated to work efficiently. In order to increase the functionality of LCNN chip, we proposed on-chip training for the LCNN chip. In this paper, we describe two training algorithms – Gradient Descent and Probabilistic Random Weight Change, which are used in LCNN on-chip training simulations. We also present the experiment results from the simulations in multi-dimensional function approximation. The training convergence is investigated and analyzed. The circuite signal flow chart for these two algorithms are designed.**

I. Introduction

The Local Cluster Neural Network (LCNN) is a special kind of Feedforward neural network proposed by Geva and Sitte [1]. It is a multilayer perceptron (MLP) where sigmoidal neurons combine in clusters that have a localised response in input space like radial basis functions (RBF). The LCNN has been demonstrated to have good performance on function approximations in digital computer simulations and it is implemented in analog VLSI hardware in the LCX chip [2] [3] [4]. Chip-in-the-loop training is successful on the LCX chip for function approximations using Probabilistic Random Weight Change (PRWC) algorithm [9]. Although chip-in-the-loop training is effective for training the analogue chip, it needs a computer and software external to the analogue chip. Based on previous research we proposed on-chip training for the LCNN. Two training algorithms are utilised for the on-chip training and they are realised by computer simulations. In this paper we describe the architecture of Local Cluster Neural Network (LCNN) briefly in section II. Followed in section III and section IV by the training methods and the training algorithms: Gradient Descent (GD) and Probabilistic Random Weight Change (PRWC) and the on-chip training strategies. Section V and section VI show the experimental results in simulations and the block circuit diagrams. The two training methods are compared in section VII. The conclusion is in section VIII.

II. Local Cluster Neural Network and its Analog Hardware Implementation

The Local Cluster Neural Network (LCNN) is defined by equation (1). Figure 1 shows the signal flow diagram for a segment of two clusters of a LCNN. The LCNN uses sigmoidal neurones in two hidden layers to form functions localised in input space, similar to Radial Basis Functions

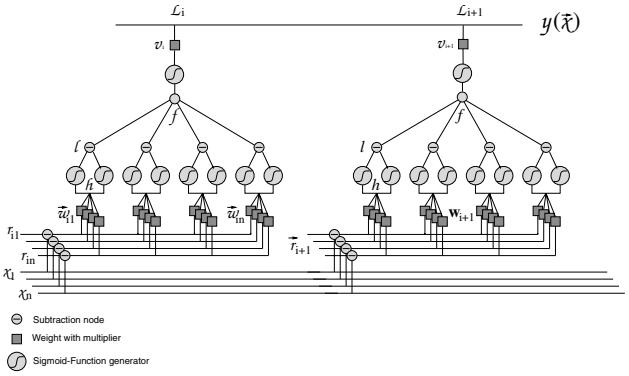

Fig. 1. Structure of the LC neural net

(RBF) but which are capable of representing a wider range of localised function shapes [3]. Each neuron in the second hidden layer outputs such a local response function. The LCNN output is a linear combination of localised scalar functions in n-dimensional input space:

$$y(\vec{x}) = \sum_{\mu=1}^{m} v_\mu L_\mu(\mathbf{W}_\mu, \vec{r}_\mu, k, \vec{x}) \tag{1}$$

where v_μ is the output weight, $\mathbf{W}_\mu$ is weight matrix for determining the output shape, $\vec{r}_\mu$ weight determines the position of the localised output function, k is the sigmoid slope and $\vec{x}$ is n-dimensional input.

The operation of LCNN as follows:

The first layer:

(i) Subtraction of the position vector $\vec{r}$ of the local function centre from the input vector $\vec{x}$ and computation of two dot products as follows:

$$h^+ = \vec{w}^T(\vec{x} - \vec{r} + 1) \tag{2}$$

$$h^- = \vec{w}^T(\vec{x} - \vec{r} - 1) \tag{3}$$

(ii) Calculation of sigmoid functions:

$$\sigma(k, h) = \frac{1}{1 + e^{-kh}} \tag{4}$$

(iii) Subtraction of the two sigmoid functions to get the ridge function (5):

$$l(\vec{w}, \vec{r}, k, \vec{x}) = \sigma(k, h^+) - \sigma(k, h^-) \tag{5}$$

The second layer:

(i) Summation of the ridge functions:

$$f(\mathbf{W}, \vec{r}, k, \vec{x}) = \sum_{i=1}^{n} l(\vec{w_i}, \vec{r_i}, k, \vec{x_i}) \tag{6}$$

(ii) Application of the sigmoidal windowing function to obtain the cluster output. Calculation of the output sigmoids as equation (7). The constant b allows shifting the window with respect to the function. Figure 2 shows a local function L in two dimensions.

$$L(\mathbf{W}, \vec{r}, k, \vec{x}) = \sigma_0(f(\mathbf{W}, \vec{r}, k, \vec{x}) - b) \tag{7}$$

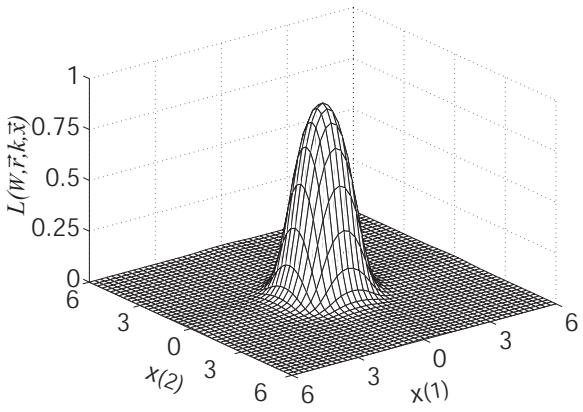

Fig. 2. Two dimensional $L(\vec{w}, \vec{r}, k, \vec{x})$

The output of the LCNN is the weighted summation of all cluster outputs.

The LCNN has been implemented in an analogue chip with 6 inputs, one output and 8 clusters [4] that can be trained by the chip-in-the-loop training scheme [9].

III. Training of Local Cluster Neural Network

Neural network training is a parameter optimisation process. Training neural networks in digital computing simulations is easy, but it is hard to realise the training in analogue neural network hardware with the same method that has utilised in digital simulations.

There are three schemes for training neural net hardware [7]. Off-chip training computes the network weights in separate computer simulation and then downloads the weight onto the chip. But this method is inaccurate, because the fluctuations and deviations in the analogue circuits are unknown and cannot be accounted for in the simulation [4] and therefore the weights obtained by training the mathematical model (LCNN) on software simulation will produce a different function on the chip. The chip-in-the-loop training scheme overcomes this problem by calculating the weights on a separate computer using the output of the analogue chip. Hence the effect caused by fluctuations and deviations is directly taken into account [9]. The ideal way of training an analogue chip is on-chip training. The on-chip training method has the training function inside the analogue chip and does not need an attached computer.

As the LCNN analogue chip does not have on-chip training circuits, in-the-loop training is applied for the LCNN chip. We formulated the Probabilistic Random Weight Change (PRWC) algorithm [9] and it is successfully used for in-the-loop training for LCNN analogue chip. Followed the research achievement of LCNN analogue hardware implementation and its in-the-loop training, we propose on-chip training for the next version of LCNN analogue chip.

IV. Training algorithms for on-chip training

We have considered two different training algorithms for on-chip training – Gradient Descent (GD) and Probabilistic Random Weight Change (PRWC).

A. Gradient Descent (GD)

The Gradient Descent (GD) has been used in the LCNN digital computing simulation with batch training as equations (8) and (9) and it has proven to perform very efficiently.

$$E = \frac{1}{p}\sum_{i=1}^{p}(y_i - y_i^*)^2 = \frac{1}{p}\sum_{i=1}^{p} e_i^2 \tag{8}$$

$$\Delta q = \eta \sum_{i=1}^{p} e_i \frac{\partial y(\vec{x_i})}{\partial q} \tag{9}$$

where p represents the training sample number, q as the parameter in the neural network architecture (LCNN has three kinds of parameters: w, r and v), y as the neural network output and y^* as the desired output.

In each training epoch, mean square error is calculated for a set of training sample points and the weight change is calculated with a sum of derivative in the set of sample points (figure 3). The learning rates are adjusted according to the sum of mean square error changes (figure 4).

This batch training can be easily realised in software but it is hard to be realised in an analogue hardware as there are many rules in the training process and many memories are needed for rules and for batch calculations. Hence it can be seen that batch training is not suitable for analogue design. Instead of batch training, we use an on-line strategy or pattern-mode for on-chip training design (figure 5).

On-line training algorithm takes the squared error in each training sample

$$e = (y_i - y_i^*)^2 \tag{10}$$

to update the weights instead of calculating mean square error from a batch of calculations. The learning rate is a fixed small value in training instead of adapted by rules. The weights are adjusted by the derivatives (equations 19 - 21) in each training sample pointer. Then the weight changes are determined by learning rate η and the error e (equations 16 - 18). Thus no rules are needed on training. This strategy simplifies the analogue training circuit compared to batch training.

The Gradient Descent (GD) algorithm in on-line training strategy is defined as equations (11) and (12):

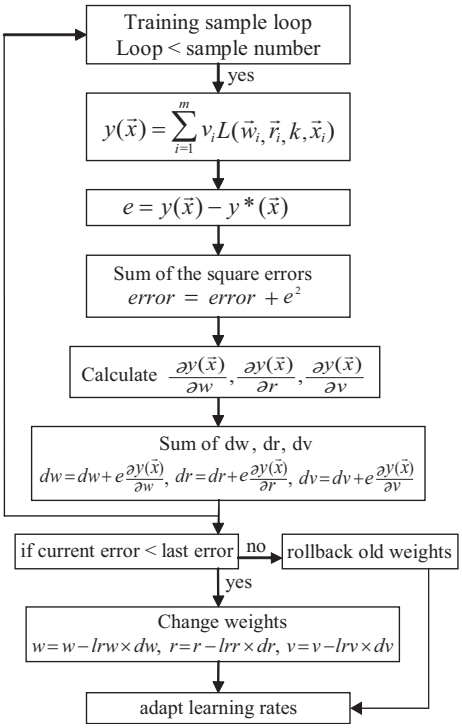

Fig. 3. The flow chart of LCNN GD batch training

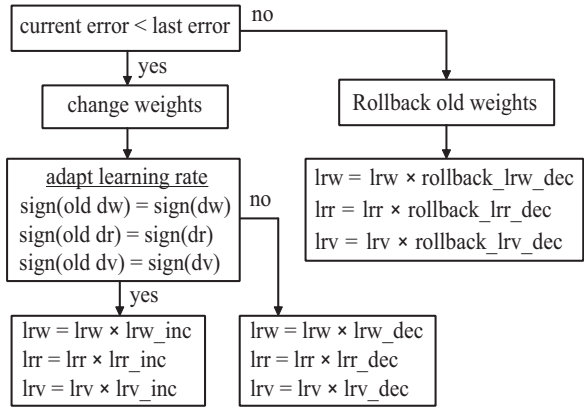

Fig. 4. Rules of adapting learning rates on LCNN training.

$$\Delta q = \eta \frac{\partial e}{\partial q} \tag{11}$$

$$q = q + \Delta q \tag{12}$$

where η is the learning rate, q is the parameter in neural network.

The LCNN includes three kinds of parameters (weights). They are updated as equations (13) - (15) in training. The weight changes are calculated in equations (16) - (21).

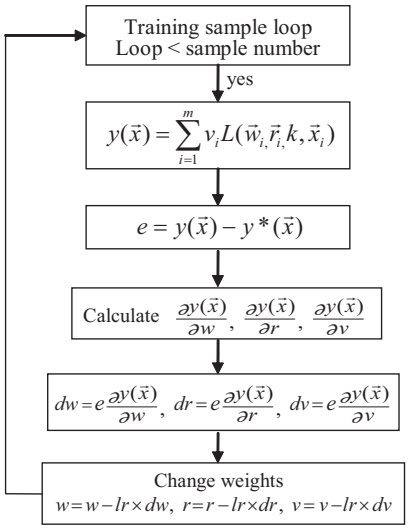

Fig. 5. The flow chart of LCNN GD on-line training

$$w_{\mu ij}(n+1) = w_{\mu ij}(n) - \Delta w_{\mu ij} \tag{13}$$
$$r_{\mu j}(n+1) = r_{\mu j}(n) - \Delta r_{\mu j} \tag{14}$$
$$v_{\mu}(n+1) = v_{\mu}(n) - \Delta v_{\mu} \tag{15}$$

$$\Delta w_{\mu ij} = \eta e \frac{\partial y(\vec{x_p})}{\partial w_{\mu ij}} \tag{16}$$
$$\Delta r_{\mu j} = \eta e \frac{\partial y(\vec{x_p})}{\partial r_{\mu j}} \tag{17}$$
$$\Delta v_{\mu} = \eta e \frac{\partial y(\vec{x_p})}{\partial v_{\mu}} \tag{18}$$

$$\frac{\partial y(\vec{x_p})}{\partial w_{\mu ij}} = v_{\mu} \sigma_o^{'}(f_{\mu}(\vec{w}, \vec{r}, k_1, \vec{x_p}) - b) \frac{\partial f_{\mu}(\vec{x_p})}{\partial w_{\mu ij}} \tag{19}$$
$$\frac{\partial y(\vec{x_p})}{\partial r_{\mu j}} = v_{\mu} \sigma_o^{'}(f_{\mu}(\vec{w}, \vec{r}, k_1, \vec{x_p}) - b) \sum_{i=1}^{n} \frac{\partial l_{\mu i}(\vec{x_p})}{\partial w_{\mu ij}} \tag{20}$$
$$\frac{\partial y(\vec{x_p})}{\partial v_{\mu}} = L_{\mu}(\vec{w}, \vec{r}, k_1, k_2, \vec{x_p}) \tag{21}$$

B. Probabilistic Random Weight Change (PRWC)

The Probabilistic Random Weight Change (PRWC) is an alternative method that we proposed for on-chip analogue design. The PRWC training algorithm is "model-free" such as Random Weight Change (RWC) [5], Weight Perturbation (WP) [6] and Simulated Annealing (SA) [8]. In the comparison with the GD on-line algorithm, the PRWC algorithm results in further simplifcation in circuit design as it does not require intermediate network outputs. PRWC has been successfully applied in the LCNN analogue chip in-the-loop training with batch strategy. We propose on-line strategy to design our on-chip training.

The PRWC is defined as follows:

$$w_i(n)' = w_i(n) + \Delta w_i(n) \tag{22}$$

where n is the number of training sample and i is the weight index. In each training sample pointer, we have the original weight set w_i and the new weight set w_i', such that we have training errors e and e' in each training sample from equation (10). The weight change Δw_i is:

$$\Delta w_i = \begin{cases} Lr & k = 0 \\ (-1) \times Lr & k = 1 \\ 0 & k \neq 0 \quad \& \quad k \neq 1 \end{cases} \tag{23}$$

$$k = rand \bmod m \qquad (m = 3, 4, 5, ...) \tag{24}$$

where Lr is the learning rate, $rand$ is a positive random value and k is the remainder of the random number divided by m.

The weight w_i and weight change Δw_i in the next training sample are decided by equations (25) and (26).

$$\begin{cases} w_i(n+1) = w_i(n)' & e(n)' < e(n) \\ w_i(n+1) = w_i(n) & e(n)' \geq e(n) \end{cases} \tag{25}$$

$$\begin{cases} \Delta w_i(n+1) = \Delta w_i(n) & e(n)' < e(n) \\ \Delta w_i(n+1) = new\_\Delta w_i & e(n)' \geq e(n) \end{cases} \tag{26}$$

In each training sample, w_i is updated by equation (22). The weight change Δw_i is determined by k in equation (23). Thus the weight w_i is randomly changed only while $k = 0$ or $k = 1$. i.e. w_i is changed by probability $2/m$. The probability $2/m$ is determined by m. If the error decreases, then the weight change Δw_i for the next training sample will keep in the same value as in the last training sample. Otherwise, if the error increases, Δw_i will be set by equation (23). Figure 6 shows the block diagram of PRWC on-chip training.

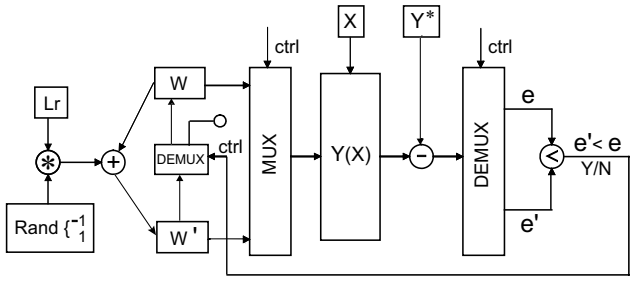

Fig. 6. Block diagram for LCNN Random Weight Change training

V. Experiment results in simulations

We have tested both GD on-line training and PRWC on-line training in simulations. In this section results from these two simulations are shown.

The Mexican Hat function (27), subtraction of two Gaussian functions (28) and sine function (29) are used as our testing functions in multi-dimensional function approximations.

$$y = a \cdot cos(c \cdot |\vec{x}|^2) \cdot e^{-\frac{|\vec{x}|^2}{b}} \tag{27}$$

$$y = a \cdot e^{-\frac{(x_1-c)^2+(x_2-c)^2}{2\times b^2}} - a \cdot e^{-\frac{(x_1+c)^2+(x_2+c)^2}{2\times b^2}} \tag{28}$$

$$y = sin(x) \tag{29}$$

where a, b and c are the parameters that determine the output shape.

We did statistical testing for the two training methods. Table I and II present the average training errors in multi-dimensional mexhat functions and their standard deviations for GD and PRWC training. Table III and IV present the average training errors in sine time series multi-dimensional data and their standard deviations for GD and PRWC training.

TABLE I

The GD training on statistic testing for mexhat function

	1-d	2-d	3-d	4-d	5-d	6-d
average error	0.0347	0.0397	0.0871	0.0482	0.0881	0.0343
standard deviation	0.0014	0.0083	0.0009	0.0010	0.0003	0.0018

TABLE II

The PRWC training on statistic testing for mexhat function

	1-d	2-d	3-d	4-d	5-d	6-d
average error	0.0376	0.0522	0.0952	0.0497	0.0968	0.0926
standard deviation	0.0083	0.0130	0.0012	0.0005	0.0016	0.0051

TABLE III

The GD training on statistic testing for sine time series data

	2-d	3-d	4-d	5-d	6-d
average error	0.0168	0.0155	0.0149	0.0148	0.0142
standard deviation	0.0032	0.0038	0.0033	0.0025	0.0029

TABLE IV

The PRWC training on statistic testing for sine time series data

	2-d	3-d	4-d	5-d	6-d
average error	0.0294	0.0195	0.0164	0.0159	0.0129
standard deviation	0.0108	0.0083	0.0026	0.0045	0.0035

The one dimensional mexhat function approximation training with 4 clusters of LCNN in GD is displayed by figures 7 and 8. After 10000 GD training epochs, the final training error is 0.035 as showing in figure 7. The plot in figure 8 shows desired output (solid line) and the GD training output

(dashed line). In comparison with GD training, the figures 9 and 10 display the PRWC training in the same training samples and with the same clusters. After 10000 PRWC training epochs, the training error is 0.029 in minimum. The plot in figure 10 shows desired output (solid line) and the PRWC training output (dashed line).

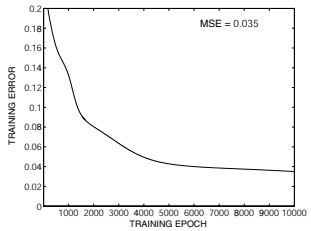

Fig. 7. The training error of one dimensional mexhat function in GD training.

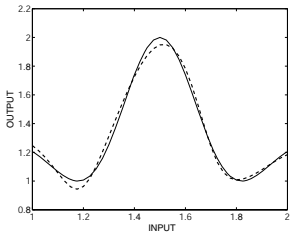

Fig. 8. The output of one dimensional mexhat function by GD training (solid line: target output, dashed line: LCNN output).

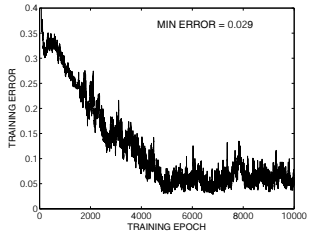

Fig. 9. The training error of one dimensional mexhat function in PRWC training

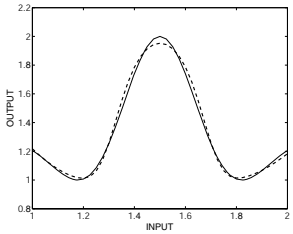

Fig. 10. The output of one dimensional mexhat function by PRWC training (solid line: target output, dashed line: LCNN output).

Figure 11 shows the training error for two dimensional mexhat function approximation with 8 clusters of LCNN in GD training. After 10000 training epochs, the training error is 0.036. Figure 12 shows the desired 2D mexhat output (left plot) and the GD training output for 2D mexhat function approximation (right plot).

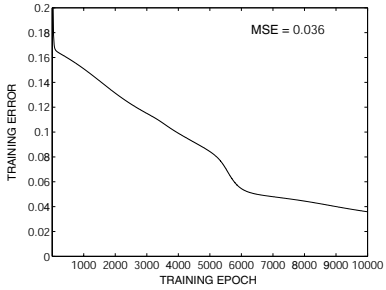

Fig. 11. The GD training error for 2 dimensional mexhat testing

The same two dimensional mexhat training sample is tested by PRWC training in 8 clusters of LCNN. Figure 13 shows the PRWC training error and figure 14 shows the desired output and the PRWC training output in 2D mexhat

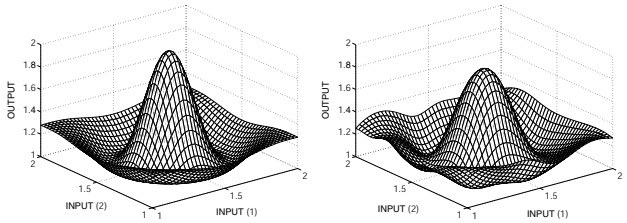

Fig. 12. The training target and output for 2 dimensional mexhat testing in GD.

function. After 10000 training epochs, the training error is 0.040.

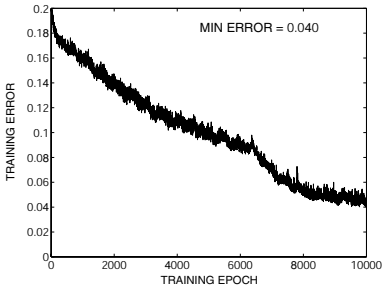

Fig. 13. The PRWC training error for 2 dimensional mexhat testing

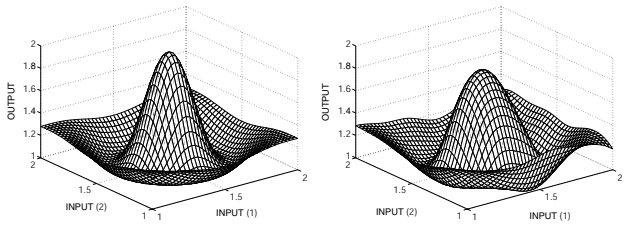

Fig. 14. The training target and output for 2 dimensional mexhat testing.

The results show that the GD training error decreases smoothly through training epoch. In PRWC training the error decreases not as smoothly as GD training because it adjusts the weights randomly but the error shows the same decreasing trend as GD. In addition, the PRWC training is faster than GD training because the PRWC involves less computation than GD. In the final hardware realisation the speed difference will be less and instead there will be a saving in circuit area.

VI. On-chip training block circuit design

Figure 15 shows the Gradient Descent (GD) on-chip training block circuit diagram. We have used derivative calculations and many multipliers to complete the training procedure in flow chart 5. Figure 16 shows the Probabilistic Random Weight Change (PRWC) on-chip training block circuit diagram. Two weight storages are needed to keep the original weights and the updated weights and fewer multipliers are needed to complete the PRWC training procedure as described in figure 6.

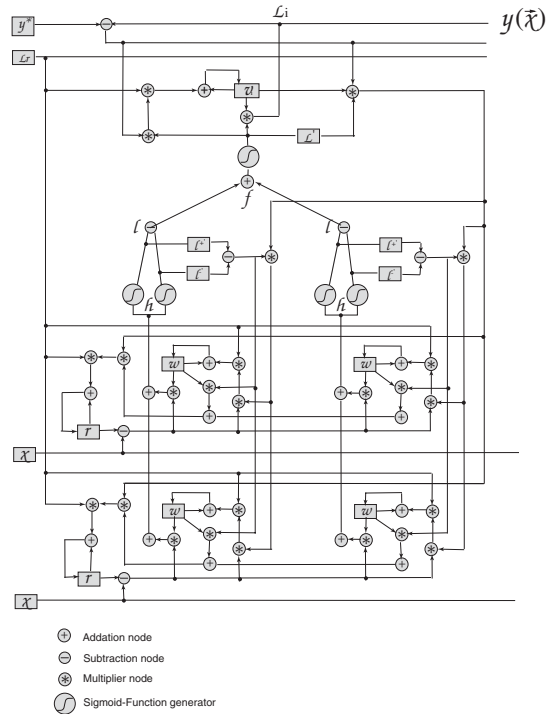

Fig. 15. LCNN Gradient Descent on-chip training block diagram

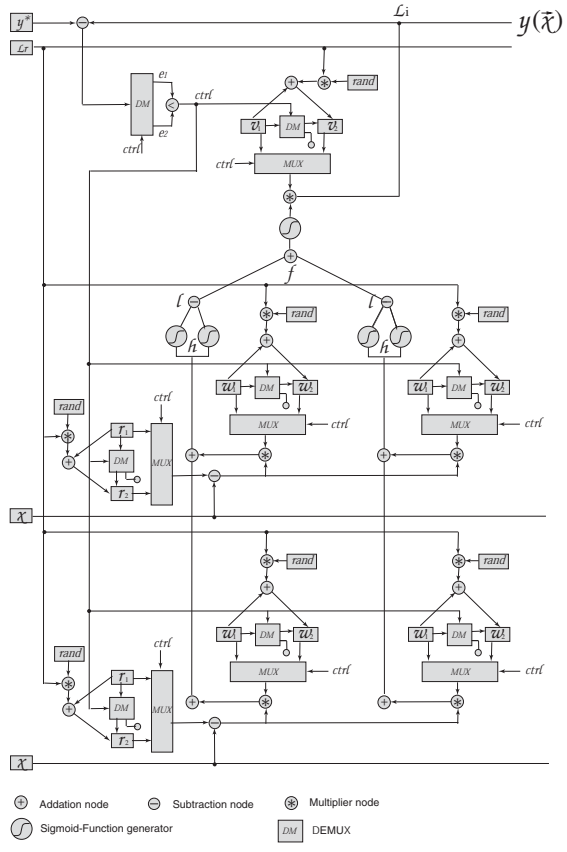

Fig. 16. LCNN Random Weight Change on-chip training block diagram

VII. Comparison of PRWC and GD

Analysing the experimental results in section V and the block circuit diagram design in section VI, we find the following characteristics for PRWC and GD training:

- Approximation error: The multi-dimensional time series experiment, two methods performan equally well. In normal multi-dimensional function approximation, the final error is affected by the type of target function and dimensionality. For complicated target functions, the GD training is more stable than the PRWC training.
- Convergence speed: The PRWC and GD have the similar convergence speed measured by the number of training epochs required to converge to the smallest error.
- Complexity: Comparing PRWC and GD block circuit diagrams, the GD needs much more computation than PRWC in training. So that GD is slower than PRWC in training processing and GD is more complicated than PRWC in fabrication.

VIII. Conclusions

The training methods, GD and PRWC, were successfully used in analogue on-chip training simulations. The PRWC is a "model-free" training method. Both algorithms are applied in on-line training strategy. Both methods performed well, statistically, in final error and convergence speed. Although the two methods have the different advantages in different cases, in overall, the PRWC training method has more potential for analogue hardware on-chip training implementation as its simplicity is suitable for achieving a high degree of parallelism on the chip. The analogue LCNN with on chip training is intended for control applications such as for example brushless DC motor control.

References

[1] S. Geva, K. Malmstrom, J. Sitte, "Local Cluster Neural Net: Architecture, Training and Applications," *Neurocomputing*, 20, pp. 35–56, Mar. 1998.

[2] Tim Körner, *Analog VLSI Implementation of a Local Cluster Neural Net*, PhD thesis, University of Paderborn, 2000.

[3] J. Sitte, T. Körner, U. Rückert, "Local Cluster Neural Net Analog VLSI Design," *Neurocomputing*, 19, pp. 185–197, Aug. 1997.

[4] L. Zhang, J. Sitte, U. Rückert, "Local Cluster Neural Network Chip for Control," *Proc. 7th International Conference on Adaptive and Natural Computing Algorithms*, Coimbra, Portugal, 2005, pp. 116–119.

[5] J. Liu, M. Brooke, K. Hirotsu, "A CMOS Feedforward Neural-Network Chip With On-Chip Parallel Learning for Oscillation Cancellation," *IEEE Transactions on Neural Networks*, vol. 13, pp. 1178–1186, Sep. 2002.

[6] M. Jabri, B.F. Flower, "Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 154–157, 1992.

[7] G. Cairns, L. Tarassenko, "Learning with Analogue VLSI MLPs," *Proc. 4th International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, 1994, pp. 67–76.

[8] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing," *Science* vol. 220, pp. 671–680, 1983.

[9] L. Zhang, J. Sitte, "Hardware In-the-loop Training of Analogue Neural Network Chip," *Proc. Third International Symposium on Neural Networks*, Chengdu, China, May 2006, pp. 1134–1139.