Multiplier-Free Feedforward Networks

Altaf H. Khan
174 Eden Avenue, Defence Road, Lahore, Pakistan (altaf@altafkhan.com)

Abstract - A feedforward network is proposed which lends itself to cost-effective implementations in digital hardware and has a fast forward-pass capability. It differs from the conventional model in restricting its synapses to the set $\{-1, 0, 1\}$ while allowing unrestricted offsets. Simulation results on the 'onset of diabetes' data set and a handwritten numeral recognition database indicate that the new network, despite having strong constraints on its synapses, has a generalization performance similar to that of its conventional counterpart.

I. Hardware Implementation

Ease of hardware implementation is the key feature that distinguishes the feedforward network from competing statistical and machine learning techniques. The most distinctive characteristic of the graph of that network is its homogeneous modularity. Because of its modular architecture, the natural implementation of this network is a parallel one, whether in software or in hardware.

The digital, electronic implementation holds considerable interest – the modular architecture of the feedforward network is well matched with VLSI design tools and therefore lends itself to cost-effective mass production. There is, however, a hitch which makes this union between the feedforward network and digital hardware far from ideal: the network parameters (weights) and its internal functions (dot product, activation functions) are inherently analog. It is too much to expect a network trained in an analog (or high-resolution digital) environment to behave satisfactorily when transplanted into typically low-resolution hardware. Use of the digital approximation of a continuous activation function, and/or range-limiting of weights should, in general, lead to an unsatisfactory approximation. The solution to this problem may lie in a bottom-up approach – instead of trying to fit a trained, but inherently analog network in digital hardware, train the network in such a way that it is suitable for direct digital implementation after training. This approach is the basis of the network proposed here. This network, with synapses from $\{-1, 0, 1\}$ and continuous offsets[1], can be formed without using a conventional multiplier. This reduction in complexity, plus the fact that all synapses require no more than a single bit each for storage[2], makes these networks very attractive.

It is possible that the severity of the $\{-1, 0, 1\}$ restriction may weaken the approximation capability of this network, however our experiments on classification tasks indicate otherwise. Comfort is also provided by a result on approximation in $C(\mathbb{R})$ [4]. That result, the Multiplier-Free Network (MFN) existence theorem, guarantees that networks with input-layer synapses from the set $\{-1, 1\}$, no output-layer synapses, unrestricted offsets, and a single hidden layer of neurons requiring only sign adjustment, addition, and hyperbolic tangent activation functions, can approximate all functions of one variable with any desired accuracy.

The constraints placed upon the network weights may result in an increase in the necessary number of hidden neurons required to achieve a given degree of accuracy on most learning tasks. It should also be noted that the hardware implementation benefits are valid only when the MFN has been trained, as the learning task still requires high-resolution arithmetic. This makes the MFN unsuitable for in-situ learning. Moreover, high-resolution offsets and activation function are required during training and for the trained network.

II. Approximation in $C(\mathbb{R})$

Consider the function $\hat{f}$:

$$\hat{f}(x) = \sum_{i=1}^{I} \sigma(a_i x + b_i) + \sum_{j=1}^{J} \psi(c_j x + d_j) + e \quad (1)$$

where $a_i, c_j \in \{-1, 1\}$, $b_i, d_j, e \in \mathbb{R}$, $\sigma(\cdot) = tanh(\cdot)$, $\psi(\cdot) = \alpha\sigma(\cdot)$, and $\alpha \in \mathbb{R}\backslash\mathbb{Q}$. The universal approximation property of $\hat{f}$ can be stated as follows [4]:

MFN Existence Theorem Finite sums of the form $\hat{f}$ are uniformly dense on compacta in $C(\mathbb{R})$. □

The network of equation 1 trades off the complexity of individual neurons with a possible increase in their number. Although the complexity of the learning algorithm is increased by the presence of two distinct activation functions in the hidden layer, it is somewhat compensated for by the lack of weights in the output layer.

This universal approximation result is for a SISO network. No similar theoretical evidence for universal approximation of multivariable continuous functions is currently available, although the author conjectures it to be true. This conjecture was tested with a series of experiments with MIMO MFNs. The MIMO MFNs used for these experiments differed with the SISO MFN of in two ways: first, they were allowed to have output layer synapses, but these synapses were restricted to $\{-1, 0, 1\}$

[1] Offsets are also known as thresholds as well as biases.
[2] A zero-valued synapse indicates the absence of a synapse!

to preserve the multiplier-free functionality; secondly, α was set equal to 1, which resulted in a homogeneous hidden layer. Neither step was necessary for the success of these experiments, but they were taken to improve the size and speed of the implementation: the first step makes unnecessary the redundant presence of two hidden neurons with outputs differing in polarity only and allows the flexibility of not connecting some hidden neurons to some of the output neurons; the second step makes unnecessary the presence of two separate types of hidden neurons, without noticeably degrading the learning performance during the experiments.

III. Learning Procedure

A procedure for training feedforward networks having discrete synapses and offsets will be presented here. It can be adapted to train networks with various discrete-weight schemes. This procedure, similar to the one proposed in [5], was used to train only the synapses of the MFNs – the offsets were trained using conventional error Back-Propagation (BP). The same procedure was used to train networks having discrete synapses as well as offsets. These Discrete-Weight Networks (DWN), having weights from the set $\{-3,-2,-1,0,1,2,3\}$, as well as the conventional continuous-weight networks (CWN), were used to get a measure of the relative generalization performance of the MFNs.

The procedure starts by initializing the network with small, continuous weights selected randomly from a uniform distribution. The conventional on-line BP procedure is used for the minimization of the output error, E_o. A weight discretization mechanism is superimposed on this BP procedure to minimize the difference between weights and a static weight discretization function, $\mathcal{Q}(w)$, in the mean-squared sense. Therefore, the combined error function is:

$$\begin{aligned} E(\mathbf{W}) &= E_o(\mathbf{W}) + E_w(\mathbf{W}) \\ &= \sum_{\substack{\text{all outputs}\\ \text{all examples}}} [t-o]^2 \; + \; \sum_{\text{all weights}} [w-\mathcal{Q}(w)]^2 \end{aligned} \quad (2)$$

The choice of discretization function is critical for the efficient discretization of weights. The discretization function used for the experiments of this paper is:[3]

$$\mathcal{Q}_{tanh}(w) = \sum_{i=0}^{3} \mathsf{tanh}[3\,(w+2i-3)] \quad (3)$$

The key feature of this function is that the zeros of $(w - \mathcal{Q}_{tanh}(w))$ have the required discrete values. In practice, the application of this function is restricted to the interval $[-1,1]$ for MFNs and $[-3,3]$ for DWNs, since any weight values outside this interval are truncated to $\{-1,1\}$ and $\{-3,3\}$, respectively.

[3]Another possible choice is $\mathcal{Q}_{sin}(w) = w + \frac{\sin(w+1)\pi}{\pi}$. Of the two discretizing functions, $\mathcal{Q}_{tanh}(w)$ is computationally more expensive to generate, yet it does have the advantage in having an adjustable slope between discrete values. It also uses the same hyperbolic tangent function as the neuron activation function.

TABLE I
Discrete-weight learning

Initialize weights in the range $(-0.5, 0.5)$
Select example
Select a weight, w
$w \leftarrow w \; - \; \eta \frac{\partial E_o}{\partial w}$
$w \leftarrow w \; - \; \chi(w - \mathcal{Q}_{tanh}(w)) \left(1 - \frac{\partial \mathcal{Q}_{tanh}(w)}{\partial w}\right)$
Loop
Loop until $E_o < \varepsilon$ and $E_w = 0$

The weight modification that can be used to minimize the error function of Equation 2 is:

$$\Delta w = -\Delta w_o \; - \; \Delta w_w \quad (4)$$

$$\Delta w_o = \eta \frac{\partial E_o}{\partial w} \quad (5)$$

$$\begin{aligned} \Delta w_w &= \chi \frac{\partial E_w}{\partial w} \\ &= \chi\big(w - \mathcal{Q}_{tanh}(w)\big)\left(1 - \frac{\partial \mathcal{Q}_{tanh}(w)}{\partial w}\right) \end{aligned} \quad (6)$$

where η and χ are the learning and weight discretization rates, respectively. A summary of the learning procedure based on these equations is shown in Table I.

This learning scheme is a modified version of the conventional on-line BP procedure – the modification being the addition of a nonlinear weight shaping function to the objective function. This modified objective function contains a combination of hyperbolic tangent functions only. As the hyperbolic tangent function is continuously differentiable, all the discussion in [6] and [10] applies. Hence, discrete-weight learning with a constant learning rate is guaranteed to converge in probability to a solution provided that the sequence of training examples is strongly stationary and ergodic [6]. Convergence with probability 1 can be achieved by using a sequence of diminishing learning rates [10].

A. Practical Considerations

The results of the initial trials with the discrete-weight learning procedure were not promising. Many of the trial runs failed to converge even on simple learning tasks. The number of epochs required for convergence to a solution was unacceptably large because the weight values hovered around discrete levels for too long without actually reaching them. This problem was due to the interaction of the error minimization and the discretization mechanisms – the two mechanisms were nullifying each other's weight modifications which was resulting in paralysis. The following guideline was drafted to resolve the problems observed during the initial trials:

> The procedure must mainly be based on gradient descent to benefit from the fast speed of that

heuristic. It should, however, have a stochastic component to account for the large number of local minima observed during the initial trials. The paralysis observed during the initial stages of learning should be avoided by letting the BP mechanism dominate the learning process when E_o is large. The weight-discretization mechanism can have the upper hand when E_o is small. This process should be augmented with a weight-rounding mechanism to speed-up convergence when weights have 'nearly-discrete' values. Lastly, the time consuming exact calculations should be replaced with their approximate but faster incarnations.

Keeping these ideas in mind, it was decided that the learning process should start with little attention to weight discretization: discretization should slowly come into play as the network starts moving towards a solution, and should gain strength with progress in learning. This was accomplished by computing a new E_o-dependent χ before each learning epoch: χ was made exponentially dependent upon the negative of E_o.

$$\chi := \alpha_\chi \, e^{(\varepsilon - E_o)\beta_\chi} \tag{7}$$

where α_χ and β_χ are empirical parameters, and ε is the maximum acceptable value of E_o. This strategy did help in the initial stage of training but not during the intermediate stage. The most likely reason was the formation of new local minima due to discretization. It is well known that the standard BP algorithm sometimes gets stuck in local minima. The superposition of the discretization process on BP can result in changes in the shapes of these minima [12] or even an increase in their number. At the start, E_o is large, and output error minimization dominates. Conversely, the discretization process has the upper hand when E_o is small. At intermediate values of E_o, the two processes may nullify each other's effect – resulting in the formation of local minima. To avoid those minima, the discretization process was augmented by a perturbation mechanism:

$$\Delta w_w \leftarrow \Delta w_w \tan(RND), \tag{8}$$

where *RND* is a random number selected uniformly from $(0, \pi/2)$. $\tan(RND)$ has a small value most of the time, but infrequently, it becomes very large. This perturbation strategy was successful, except in cases where a discrete-weight network with an unacceptably large E_o was generated. The solution that was adopted in those cases was to strengthen the E_o minimization process. This was accomplished by temporarily boosting the value of η by a factor of k_η for one learning epoch.

Weight-discretization is based on a mean-squared error minimization heuristic. Its progress becomes painfully slow as the system becomes closer to a solution. To overcome this problem, a rounding mechanism for weights with 'nearly-discrete' values was added to the discretization process. This mechanism acts as a set of 'black holes' centered at each discrete value. If a weight falls within the black hole radius, ρ, its value is forced to the center value of the black hole. The radius was computed before each learning epoch, and was made exponentially dependent upon the negative of E_o.

$$\rho := \alpha_\rho \, e^{(\varepsilon - E_o)\beta_\rho} \tag{9}$$

where α_ρ and β_ρ are empirical constants. The black-hole function, $\mathcal{B}(w)$, for the DWN case can now be defined:

$$\mathcal{B}(w) = \begin{cases} 3\,\mathsf{sgn}[w] & |w| > 3, \\ \mathsf{Round}[w] & |w - \mathsf{Round}[w]| < \rho, \\ w & \text{otherwise.} \end{cases} \tag{10}$$

Finally, the $(w - \mathcal{Q}_{tanh}(w))\left(1 - \frac{\partial \mathcal{Q}_{tanh}(w)}{\partial w}\right)$ term was replaced by an easily computed approximation $(w - \mathrm{Round}[w])$. Easier computation is not the only advantage of the approximate term – it also provides stronger 'pull' when a weight value is midway between two discrete levels. After making all of the changes the final version of the Δw_w modification is:

$$\Delta w_w = \alpha_\chi \, e^{(\varepsilon - E_o)\beta_\chi} (w - \mathsf{Round}[w]) \tan(RND). \tag{11}$$

A momentum term was added to the BP mechanism to strengthen the E_o minimization process.

$$\Delta w_o = \eta \frac{\partial E_o}{\partial w} \; - \; \mu \Delta w_{o_{previous}} \tag{12}$$

where μ is the momentum and $\Delta w_{o_{previous}}$ is the Δw_o calculated in the previous step. A summary of this more practical version of the discrete-weight learning procedure is shown in Table II.

TABLE II
PRACTICAL DISCRETE-WEIGHT LEARNING

Initialize weights in the range $(-0.5, 0.5)$
If $E_w = 0$ and $E_o > \varepsilon$
  then boost η by a factor of k_η for just this epoch
Select training example
  Select a weight, w
    $w \leftarrow w \; - \; \eta \frac{\partial E_o}{\partial w} \; + \; \mu \Delta w_{o_{previous}}$
    $w \leftarrow w \; - \alpha_\chi e^{(\varepsilon - E_o)\beta_\chi} (w - \mathsf{Round}[w]) \tan(RND)$
    $w \leftarrow \mathcal{B}(w)$
  Loop
Loop until $E_o < \varepsilon$ and $E_w = 0$

B. Functionality Tests

A set of three learning tasks was used for testing the functionality of the proposed learning procedure – XOR and two encoder/decoder problems, 4:2:4 and 8:3:8. Feedforward networks with offsets and hyperbolic tangent activation function in both the hidden and output layer neurons were used for these simulations. The training data was scaled to the range $[-1, 1]$. Functionality testing involved clean data therefore the L_∞-norm, E_{o_m}, was used as the error function: training was stopped when E_{o_m} was less than a prespecified ε and all the weights had reached discrete values. The network was reinitialized

TABLE III
VALUES OF WEIGHT DISCRETIZATION PARAMETERS

Parameter	Symbol	α	β
Weight-discretization rate	χ	0.001	16
Black-hole radius	ρ	0.1	6

if it did not converge to a satisfactory solution within a fixed number of epochs, C_R. These tasks were used to monitor the behavior and interactions of the various mechanisms present in the learning procedure and for the fine-tuning of the learning parameters.[4] The selected values for these parameters are shown in Table III.

IV. Generalization Performance

The synapses of MFNs, and synapses as well as offsets of DWNs, have limited magnitudes. Moreover, these networks usually have larger hidden layers, compared with their CWN counterparts. Because of the larger number of neurons in their hidden layers, their complexity is more finely selectable than that of CWNs. This finer granularity can be exploited to select a network with a complexity that matches more closely with the complexity of the learning task at hand. It was found from the preliminary simulations on DWNs and MFNs that some of the weights of the trained DWNs and MFNs had zero values. That reduced the number of superfluous parameters in the model. These three factors – smaller weights, a more finely granular complexity, and zero weights – point towards a simpler model, which should result in DWNs and MFNs having a generalization performance better than or equal to that of the CWNs.

The methodology of the generalization experiments presented here was to train many CWNs, DWNs and MFNs on a given set of data in similar circumstances, and then to pick as representative of each of the three paradigms the network which was the best generalizer, and compare the performances of those representatives. Two data sets were used for these experiments. Both – the 'onset of diabetes prediction' data set and the handwritten numeral recognition database – are publicly available and have been used in the past by many for the purpose of benchmarking [2], [3], [7], [9]. The first set has both discrete and continuous inputs, may have some irrelevant inputs, may have much noise in the inputs, may have a high degree of correlation between inputs, and has a single binary output [7], [9]. The second set is a representative of the image processing applications, has a high degree of information redundancy, has discrete inputs, and many binary outputs [2], [3].

The regularization method selected for the experiments was 'optimal stopping of training'. It is fast, works well with larger-than-necessary initial networks, and does not require the introduction of any new training parameters except the ratio of train/test data split. The CWN experiments presented here use weight-decay in addition to 'optimal stopping of training.' This makes the comparison between DWNs and MFNs, and CWNs more meaningful, because DWNs and MFNs are not allowed to have large weight values.

The *only* goal of the experiments presented in this section is to compare the learning capability of the three paradigms, CWN, DWN and MFN, in similar circumstances. Keeping this in mind, the 'classification error probability on the test data' was used as the metric of comparison, not sensitivity or specificity, as the goal is to compare learning capability of three paradigms and not the usefulness of the model.

The train-and-test technique was chosen to estimate the generalization performance because of its simplicity of implementation. Only two, instead of the recommended three, data subsets were, however, used for this purpose – the first one for training, and the second one for optimal stopping of training as well as estimating the generalization performance. The dual use of the second subset is not strictly appropriate as it has been used during training and therefore the performance metric extracted from it is not an unbiased estimate of the generalization performance of the trained network. In the case of the experiments discussed here, however, the emphasis is on comparing the relative generalization performances of the three paradigms in similar circumstances, and not estimating their *true* generalization performance. This particular emphasis on comparison was the main reason for the use of just two data subsets.

Network configurations with zero hidden neurons were tested first, and then hidden neurons were added until the network performance on test data failed to improve. At least a hundred training runs were performed for each network configuration, and many hundreds for promising ones. Test data results reported here represent the best performance of the optimal configurations.

A. Forecasting the Onset of Diabetes

This data set[5] is related to a group of adult women belonging to the Pima Indian tribe and was collected by the US National Institute of Diabetes and Digestive and Kidney Diseases [1]. The learning task is to forecast the onset of diabetes mellitus within five years of a clinical examination. Eight risk factors, which were recorded

[4]The fine-tuning was made tricky by the presence of many adjustable parameters. An exhaustive study of the effect of variations was not attempted because of the sheer magnitude of the task. Instead, the values of the weight-discretization parameters were frozen after some initial experiments and only the BP learning rate η was used as the control parameter. The values of μ and ε were also kept constant for these simulations and were varied only for the generalization experiments presented latter in this paper.

[5]The data set is available at `ftp://ics.uci.edu/pub/-machine-learning-databases/pima-indians-diabetes/` [8]

TABLE IV
GENERALIZATION PERFORMANCE ON THE DIABETES DATA

Network	Config.	Zero Weights	Effective Weights	Gen. Perf.
CWN	8:2:1	–	21	78.4%
DWN	8:6:1	26	35	76.9%
MFN	8:3:1	6	25	78.0%

TABLE V
GENERALIZATION PERFORMANCE ON THE NUMERAL DATA

Network	Config.	Zero Weights	Effective Weights	Gen. Perf.
CWN	32:7:10	–	311	92.3%
DWN	32:10:10	169	271	91.8%
MFN	32:11:10	155	328	93.2%

during that clinical examination, are used to make this prediction. 768 such clinical histories constitute the data set out of which 268 are for patients who tested positive for diabetes within five years of the clinical examination and the rest were found to be healthy.

49% of the 768 clinical records had zero values for attributes which cannot be zero. These are most probably missing values. Moreover, it should be noted that this database may be very noisy: some of the attributes may be unimportant, some attributes may have been measured incorrectly, some of the most important attributes may have been completely missing, and one of the attributes, the diabetes pedigree function, is based on the heuristic combination of many pieces of information.

Michie et al. used a set of 22 machine-learning, neural and statistical techniques to analyze this data [7]. They used 12-fold cross-validation to determine the generalization performance and achieved their best result of 87.7% with logistic discriminant analysis, whereas a figure of 75.2% was obtained with a CWN trained using BP. Ripley [9] used 200 cases for training and 332 for testing, and ignored the rest because of missing values. The best CWN results in this study were obtained with zero hidden neurons. The best result of this study was a generalization performance of 81% obtained by using a mixture representation and the EM algorithm.

For the simulation results presented here, a balanced data set, i.e. a set having equal number of positive and negative cases, of 536 cases was randomly selected. This set was then split into two equal, balanced subsets for training and testing. All eight attributes were standardized to zero mean and unit variance. All attribute values outside the $[-1, 1]$ range were truncated to $\{-1, 1\}$

A comparison of performances is shown in Table IV. One expects DWN to be the worst performer, but why is the MFN, with its smooth and simple mappings due to the presence of weights with small magnitudes and many zero weights, not performing better or equal to the CWN? Is it because the universal approximation conjecture about its capabilities is wrong? That is a possible reason. Another possible reason may have to do with a fault on account of the experimental procedure: it is possible that the two hyperplanes drawn by the hidden neurons of the CWN are positioned in such a way that it requires the superpositioning of a large number of MFN hidden neurons to emulate them. The MFN experiments were performed with a maximum of 19 hidden neurons and the best solution that was found used only a fraction of those 19. It is possible that with more hidden neurons a better solution can be found, but the price of that tiny increase in accuracy is an impractically large hidden layer.

B. Handwritten Numeral Recognition

The numeral database[6] used for these simulations was collected at Bell Labs [3]. It consists of 1200 isolated handwritten samples of numerals 0..9. Twelve individuals were asked to provide 100 samples each while following a given writing style, resulting in 120 examples of each numeral. The sample style given to the writers was similar to the one required for the US Internal Revenue Service's machine-readable tax form, 1040EZ. The raw images of those samples were normalized and then thresholded to fit a 16×16 binary pixel grid. This database was then carved into two subsets of 600 samples each – the first five samples of each numeral from every writer were used for training and the rest for testing. Guyon et al. [3], and Druker and Le Cun [2], both have reported a generalization performance of 97% with a 256:20:10 network trained using conventional BP, and a 256:40:10 network trained using 'double backpropagation', respectively. For the simulations presented here, the 256 element matrices were transformed into 32 element vectors by summing the rows and columns. Although this 8-fold reduction in input dimension did cause a 4% reduction in generalization performance (compared with [2], [3]), it made the running of many more simulations possible due to the reduced memory and CPU requirements. Each element of these 32-element vectors was then transformed as

$$x \longleftarrow \frac{x - 8}{8}$$

The classification decisions were taken according to the neuron with the maximum signal in the output layer.

CWNs, DWNs and MFNs with various 32:q:10 configurations were trained and the results for the best networks are shown in Table V. The lack of the universal approximation property in a DWN[7] may be the cause of the slightly poor performance of the DWN, whereas the finer control over network complexity may have caused the

[6]This database has been kindly made available by Isabelle Guyon at `ftp://hope.caltech.edu/pub/mackay/data/att.database`

[7]The limitations imposed by bounded discrete weights are enough to destroy its universal approximation capability.

slightly better performance of the MFN. Nevertheless, it can be concluded that the very strong constraints placed on the weights of DWNs and MFNs have not significantly hampered their performance.

The number of zero weights in the best performing DWN and MFN is quite large, 38% and 32%, respectively. There are only 271 non-zero weights in the case of the DWN. As each of these weights is 3 bits deep, the complexity of the network is 817 bits. This means that the network requires approximately 82 bits to store the non-parametric characteristics of each numeral to give a generalization performance of about 92%. In the case of the MFN, 307 single bit synapses and 21 high-resolution offsets are needed. This leads to a figure of 64 or 81 bits per numeral depending upon the fixed-point offset resolution of 16 or 24 bits. The equivalent figures for the CWN are 498 or 746 bits per numeral. Therefore, for similar generalization performances, the ranking in terms of storage efficiency is MFN, DWN and CWN, with MFN being the best.

The DWN lacks the universal approximation capability, whereas it was conjectured in Section II that the MIMO MFN is a universal approximator. This suggests that the learning ability, and therefore the generalization performance, of the former should not be any better than that of the latter. The experimental results on both benchmarks support this suggestion. This criterion also suggests that the generalization performances of the MFN and CWN should be the same. The experiments were inconclusive in confirming this assertion – on the very noisy 'forecasting the onset of diabetes' task the CWN was better by 0.4%, and on the larger but less noisy handwritten numeral recognition MFN was better by 0.9%. These differences, and those with respect to the DWN, are nevertheless too small to manifest the supremacy of any one of the three paradigms.

V. Conclusions

The number of multiplication operations required for a forward-pass is equal to the number of non-zero synapses, $S_{\neq 0}$, in a network. Experimental results presented in this paper show that CWN and MFN require similar number of non-zero synapses to achieve similar performances. To achieve the fastest forward-pass, the CWN requires $O(n^2)$ transistors to implement each n-bit fixed-point flash-multiplication operation, whereas the MFN requires $O(1)$ transistors for a single 1-bit conditional sign-changer. It can be concluded that for similar execution speeds, the cost of implementing the multiplication operations in a CWN is $O(n^2 S_{\neq 0})$ compared with $O(S_{\neq 0})$ for the MFN. Therefore, the replacement of the multiplication operations with sign-adjustments can greatly speed up the calculations: in software and in hardware, on sequential systems and on parallel ones. Although, MFNs are, in general, larger than their conventional counterparts in terms of the number of hidden neurons, they should be more compact in hardware due to the absence of the conventional multipliers and 1-bit synapses.

Experiments to compare the generalization abilities of three types of feedforward networks – CWN, DWN and MFN – were performed in similar circumstances. The results indicate that the MFN, despite having strong constraints on its synapses, has a generalization performance similar to that of its conventional counterpart. In an application where the number of learning epochs is not of consequence, and the size and speed of the hardware implementation are the critical factors, this new network with constrained synapses holds a clear advantage over the conventional network due to the storage efficiency of its weights and the elegance of the way it implements its internal multiplication operation.

Acknowledgments: The author is grateful to Professors R. Wilson and D. Whitehouse for their guidance during the course of this work performed at the University of Warwick, England, and supported by the Commonwealth Scholarship Commission in the UK and the University of Engineering & Technology, Lahore, Pakistan.

References

[1] P. H. Bennett, T. A. Burch, and M. Miller. Diabetes mellitus in American (Pima) Indians. *Lancet*, 2:125–128, 1971.

[2] H. Drucker and Y. Le Cun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3:991–997, 1992.

[3] I. Guyon, I. Poujaud, L. Personnaz, G. Dreyfuss, J. Denker, and Y. Le Cun. Comparing different neural network architectures for classifying handwritten digits. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 2, pages 127–132, Washington, DC, 1989. IEEE Press, New York, NY.

[4] A. H. Khan. Multiplier-free feedforward neural network is a universal approximater in $C(\mathbb{R})$. In *1997 IEEE (Pakistan Section) Second National Multi Topic Conference (INMIC'97)*, pages 33–36, Islamabad, Pakistan, April 1997.

[5] A. H. Khan and E. L. Hines. Integer-weight neural nets. *Electronics Letters*, 30(15):1237–1238, July 1994.

[6] C. M. Kuan and K. Hornik. Convergence of learning algorithms with constant learning rates. *IEEE Transactions on Neural Networks*, 2(5):484–489, 1991.

[7] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, New York, 1994.

[8] P. M. Murphy and D. W. Aha. UCI Repository of Machine Learning. Department of Information and Computer Science, University of California, Irvine, CA, 1995.

[9] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, England, 1996.

[10] H. White. Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1(4):425–464, 1989. Reprinted in [11].

[11] H. White. *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell, Oxford, England, 1992.

[12] Y. Xie and M. A. Jabri. Training algorithms for limited precision feedforward neural nets. SEDAL technical report 1991-8-3, Department of Electrical Engineering, University of Sydney, NSW 2006, Australia, 1991.