AN FPGA-BASED EIGENFILTER USING FAST HEBBIAN LEARNING

K.P. Lam and S.T. Mak

Department of Systems Engineering & Engineering Management
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

ABSTRACT

We present a high-gain, multiple learning/ decay rate, "cooling off" annealing strategy to a modified Generalized Hebbian Algorithm (GHA) that gives good approximate solution within one training epoch, and with fast convergence to accurate principal components within a few more epochs. A novel bit-shifting normalization procedure is shown to bound the weight vector norm effectively and eliminates the need for performing division. This leads to an FPGA-based computational framework using only fixed point arithmetic instead of more complicated floating point design. Simulation results on Xilinx DSP System Generator tool indicate the practicality of the approach, where real-time eigenfilter can be readily implemented on field programmable gate arrays with limited resources.

1. INTRODUCTION

Recently, eigenfilters have received wide interest in adaptive design, and in different applications including smart antenna, finite impulse response (FIR) filter, and channel equalisation. For most of the systems, numerically intensive matrix factorisation computations are required for calculating eigenvalues with high precision. While such real-time computations can often be met with today's highly efficient digital signal processor (DSP), simpler procedures are needed in less critical situation by using limited resources to give acceptable solution and performance. Based on the Hebbian learning network, this paper addresses the use of only simple matrix multiplication and addition for eigenfilter construction.

Many gradient-based neural networks use a small, constant learning rate for simplicity, and rely on a measurable model difference (as from the "teacher") to drive the weight change to zero for convergence. Non-supervisory neural networks, such as the Hebbian network, do not possess this explicit evaluation feedback mechanism and hence require some internal self-organization for the learning process. The Generalized Hebbian Algorithm (GHA) is known to find the principal components based on the work of [1] and [2]. Oja's maximum *eigenfilter* is derived from consideration of a small learning rate η to the basic single-neuron Hebbian learning, given as

$$y(t) = w^T(t)x(t) \tag{1}$$

$$\begin{aligned} dw(t+1) &= w(t+1) - w(t) && (2)\\ &= \frac{w(t) + \eta y(t)x(t)}{\|w(t) + \eta y(t)x(t)\|} - w(t) && (3)\\ &\approx \eta y(t)(x(t) - y(t)w(t)) && (4)\\ &\quad \text{when } \eta \text{ is small} \end{aligned}$$

where $x(t)$ is the m-dimensional input vector and $w(t)$ is the m-dimensional weight vector. Sanger's extended work on the GHA is significant in giving a unified treatment on obtaining all the eigenvectors using a generally decreasing learning rate for a single layer of m neurons. The previous learning equation now extends to

$$\begin{aligned} y(t) &= W^T(t)x(t) && (5)\\ dW(t+1) &= \eta(t)\{y(t)x^T(t) \\ &\qquad - LT[y(t)y^T(t)]W\} && (6) \end{aligned}$$

where W is the $m \times m$ weight matrix $(= [w_1, w_2, \ldots, w_m])$, and $LT[.]$ stands for the lower triangular matrix of $[.]$.

As detailed in [3], the convergence proof establishes the strong global "attractors" of the eigenvectors, although the existence of "two" global attractors for each eigenvector (itself and its negative counterpart) is not explicitly stated. In section 2 we extend the basic GHA with an annealing procedure to give fast convergence.

However, the importance of two constraints for this fast *eigenfilter* is often overlooked: one is the explicit normalization of the weight vectors during each learning step; the other is the use of constant learning/ decay rates within the same training epoch. The first bounds all weight trajectories

Funding support from the Hong Kong Research Grants Council is gratefully acknowledged

onto the surface of a hypersphere. The second provides the necessary high gain for learning uniformly from the input, and results in a useful chaotic regularity given a fixed order of input data.

2. A MODIFIED GHA WITH ANNEALING

We generalize the learning/ decay rate from a scalar $\eta(t)$ to diagonal matrices $A(t) = [\alpha_i(t)]$, $B(t) = [\beta_i(t)]$, for $i = 1, ..., m$, in order to take account of the different attraction of the eigenvectors.

$$\begin{aligned} dW(t+1) &= A(t)y(t)x^T(t) \\ &\quad - B(t)LT[y(t)y^T(t)]W \end{aligned} \tag{7}$$

The maximal eigenvector has a very strong attraction while the others are substantially weaker. By assigning a different learning rate to each neuron, especially with much greater values to the weaker neurons, the convergence of the network can be more synchronized with $\alpha_m > \alpha_{m-1} > \alpha_1$ and/or $\beta_m > \beta_{m-1} > \beta_1$.

2.1. A high-gain approach

A high-gain approach is needed to approach the global attractors through bypassing most of the local minima. As the magnitude information of the eigenvector is not relevant, a normalization step is taken to constrain the weight vector in each learning step. This will retain the input variance information without an unnecessary growth of the magnitude of the weight vector. Although the basic GHA will automatically constrain the weight vector in the long run, it is intended to constrain it more explicitly in the high-gain approach without much increase in the computational load.

2.2. An annealing strategy

An annealing strategy is required to get the precision in the converged solution. A high-gain approach can often allow us to go to a region close to the global attractor quickly. However, it also brings in very large input excitation to the change (dw_i) of the weight vector. This will affect the accuracy of the estimated eigenvectors. A rough estimation of the Euclidean norm of dw_1 for the maximal eigenvector can be readily shown to be

$$(\|dw_1(t+1)\|_2)^2 = k(\|x(t)\|_2)^2(\alpha_1 + \beta_1) \tag{8}$$

Various "cooling-off" annealing strategies can be imposed on the time-varying learning rate and decay rate, in order to reduce the magnitude of the input excitation and bring in the necessary accuracy to the estimated values.

2.3. Example on stock data

We process a 100-day stock price data $p(t)$ to obtain the 5-day return series $r(t) = \log(p(t)/p(t-5))$. Then a three-neuron Hebbian network is used to process the three-input data of $x(t) = [r(t-1), r(t-2), (r-3)]^T$ and generates three outputs of $y(t) = W(t)x(t)$, where $W(t)$ is the estimated weight matrix for the Hebbian network. 100 samples of $x(t)$ are used for testing the 3-neuron network. The transformed outputs are then used by a Radial-basis-function network for estimating a model for $r(t)$, which is subsequently used for both in-sample and out-of-sample price prediction. Here we focus only on the first part concerning the Hebbian network for the transformed output $y(t) = W(t)x(t)$.

2.4. Problem of using a small constant learning rate

Getting stuck in local minima is a major problem as the network does not have the necessary momentum to go over them to reach the global attractors. It is not uncommon that a small learning rate of 0.001 runs into over 4,000 epochs for the necessary convergence; and sometimes even a local minimum is reached instead of a global one. In almost all cases the norm of the weight change (dw_1) for the maximal eigenvector still fluctuates within a small range. The second and third eigenvectors are still not completely converged after 4,000 epochs. Incorrect local solutions are possible even under such long training period.

2.5. Fast convergence through annealing

We intend to use as few epochs as possible to reach the global attractors by using a large learning and decay rates ($A = B =$ diag([50, 300, 500]), and a linearly decreasing annealing procedure of changing the A, B matrices to diag([0.001, 0.01, 0.01]) at the 20^{th} epoch. The global attractors are almost reached after about 2 to 3 epochs as observed in the Euclidean difference norm of the three eigenvectors. The transformed outputs also show an exact match with that obtained by using the correct eigenvectors.

3. NORMALIZATION SCHEMES

As previously discussed, one important aspect that is often overlooked in GHA is the normalization step. Although not explicitly required in the basic algorithm, such procedure is necessary to bound the magnitude of the learned weight vector without its growing to exceedingly large value that may cause numerical problems. This is especially important for our approach during the initial stage when a very high gain is applied. One simple scheme of normalization is to find the Euclidean norm $\|w_i(t)\|_2$ of the weight vector, and self-divided by this value in each iterative step for normalizing to unit magnitude. Hence, the trajectory of the weight vector is confined to the surface of a unit magnitude hypersphere.

However, the essential information of an eigenvector is contained in its direction and not in its magnitude. If the GHA effectively learns the eigenvector information, then other normalizaton schemes can be applied to result in the same directional convergence point. Thus, GHA convergence does not necessarily imply converging to the unit eigenvectors, but to the directions of the eigenvectors instead. In the following, we confine the variation of the norm to several types: (i) a *scaled* Euclidean norm, as $\gamma||w_i(t)||_2$ where γ is a scaling constant; (ii) an infinity norm, as $||w_i(t)||_\infty$; and (iii) a bit-shifting infinity norm defined as

$$||w_i(t)||_b = 2^{\lfloor \log_2 \max_j \{|w_{i,j}(t)|\} \rfloor} \qquad (9)$$

where $1 \leq j \leq m$, and $\lfloor \cdots \rfloor$ defines the integer less than or equal to the contained argument. Notice that $\max_j\{|w_{i,j}(t)|\}$ is the infinity norm $||w_i(t)||_\infty$. The use of normalization using (iii) then constrains $w_i(t)/||w_i(t)||_b$ with a lower bound of the infinity norm. The calculation of (iii) is quite straightforward using digital logic, and the normalization step can be carried out by bit-shifting without any division. This is particularly useful for FPGA implementation (as will be discussed in the following section) with fixed point arithmetic and limited resources.

3.1. Scaled Euclidean norm

In the study of evaluating (i) where the magnitude of $w_i(t)$ is *scaled* as $1/\gamma$, we notice that when γ increases from 1, the GHA has increasing difficulty to reach the correct directional convergence point (with the exception of the first eigenvector). However, when γ decreases from 1, correct directional convergence is maintained as with the case of $\gamma = 1$. This finding is interesting as it supports our earlier postulation that the GHA can effectively learn the same, correct directional information of the eigenvectors for different scaled magnitude of $w_i(t)$.

3.2. Infinity norm

In contrast against (i), the use of infinity norm normalization in (ii) does not confine the weight vector $w_i(t)$ on a hyperspherical surface. A simulation run of 20 epochs using a shorter data length of 30 was attempted. As shown in Fig. 1, the Euclidean norms $||w_i(t)||_2$ tend to oscillate at high-gain period and finally settle to non-unity levels. However, correct directional convergence is obtained despite the fluctuations in Euclidean norms, indicating that the GHA is effective in learning directional information of the eigenvectors.

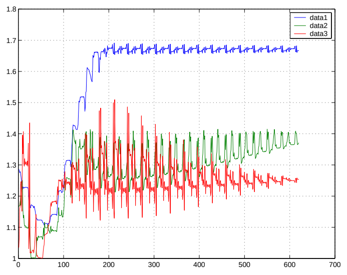

Fig. 1. $||w_i(t)||_2$ when normalized using infinity norm

3.3. Bit-shifting infinity norm

Using the same simulation data as in the infinity norm case, we evaluate the bit-shifting infinity norm normalization procedure (Eqn. (9)). It is interesting to note that, as illustrated in Fig. 2, the oscillations in the Euclidean norm $||w_i(t)||_2$ are much less violent than in (ii), and settle eventually very close to unity (despite no explicit normalization based on Euclidean norm). Fig. 3 demonstrates good directional convergence to the correct eigenvectors.

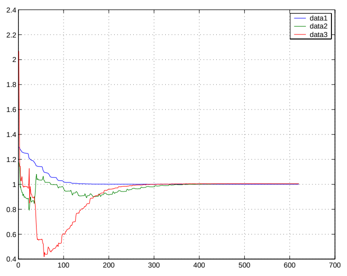

Fig. 2. $||w_i(t)||_2$ when $w_i(t)$ is normalized using bit-shifting infinity norm

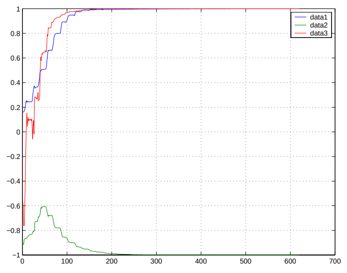

Fig. 3. Dot products between $w_i(t)/||w_i(t)||_2$ and the correct eigenvectors when $w_i(t)$ is normalized using bit-shifting infinity norm

4. FPGA COMPUTATIONAL ARCHITECTURE

Over the past decade field programmable gate arrays (FPGA) have emerged as powerful reconfigurable computing engines for many applications. In addition to the provision of a very flexible connection network for millions of logic gates on a single FPGA chip, the recent availability of high-level tools, such as Xilinx's System Generator [4], overcomes much of the tedious bottleneck of gate-level details with a user-friendly Simulink design environment of DSP blocksets. However, the current FPGA technology still suffers from a lack of powerful floating point support; and some intensive computing needs (such as division) may also cause a serious drain on the available gate resources. Our design of an FPGA-based eigenfilter takes account of these practical constraints, by realizing GHA computation with only fixed point arithmetic and a no-division normalization procedure.

The general FPGA architecture[1] of our Hebbian eigenfilter is shown in Fig. 4. While the current design is targeted on 3x3 matrix calculation, the conceptual framework can readily be extended to a more general m-dimensional eigenfilter. In realizing the required bit-shifting normalization (see Eqn. (9)), four blocks are constructed: ToPositive, MaxValue, Maxlis, and DisAmplify.

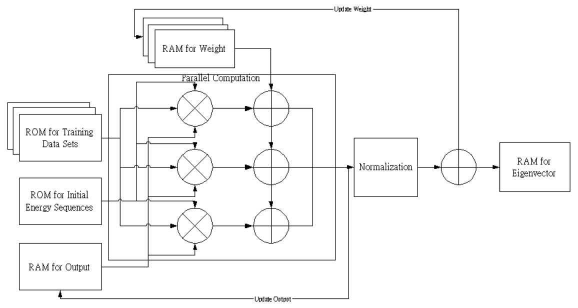

Fig. 4. General FPGA architecture for hardware GHA (HGHA)

ToPositive makes all values to positive and records their sign, and restores their sign after normalization; MaxValue finds the max value of the vector; Maxlis finds the most MSB bit of the max value ($\lfloor \log_2 \max_j\{|w_{i,j}(t)|\}\rfloor$); and DisAmplify does the shifting when the MSB value is greater than zero.

4.1. Implementation using fixed point arithmetic

Our FPGA architecture was designed using the fixed point arithmetic blocksets of Xilinx's System Generator under the Simulink environment. The software GHA (SGHA) implements Eqn. (9) with the usual floating point precision of MATLAB. In constrast, the hardware GHA (HGHA) must suffer from truncation errors due to finite bit length in fixed point number representation. Using a 64-bit length with different binary point positions, several combinations were tested: 32/32, 24/40, and 40/24. The 40/24 combination gives the best results, indicating that 40 bits are quite adequate to minimize truncation error due to large values. Fig. 5 tabulates a comparative evaluation between the SGHA and HGHA based on the error variances. The results demonstrates that our FPGA-based Hebbian eigenfilter can learn the eigenvectors reasonably well.

[1]We thank Winston Wong for his work in constructing the early design prototype of the FPGA-based eigenfilter

	Matlab error	FPGA error	Ratio
5 epochs	7.8185e-008	4.1566e-007	5.32
10 epochs	3.578e-8	1.1545e-007	3.23
20 epochs	2.6294e-8	2.7388e-007	10.41
30 epochs	1.427e-8	1.1582e-007	8.12
40 epochs	8.4330e-9	5.8224e-008	6.9

Fig. 5. Evaluating SGHA and HGHA

5. CONCLUSIONS

The normalization step of a modified generalized Hebbian algorithm (GHA) based on high-gain and annealing has been studied, with respect to both the Euclidean norm and the infinity norm. A bit-shifting infinity norm normalization procedure is successfully derived, showing good convergence property and providing the basis for an FPGA-based design. Simulation results show very good performances for the hardware GHA (HGHA) using only fixed point arithmetic of the FPGA.

6. REFERENCES

[1] E. Oja, "Neural networks, principal components, and subspaces," *Internal Journal of Neural Systems*, vol. 1, pp. 61–68, 1989.

[2] T.D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, vol. 12, pp. 459–473, 1989.

[3] S. Haykin, *Neural networks: A comprehensive foundation*, New York : Macmillan College Publishing Company, 1994.

[4] Xilinx, *Xilinx System Generator for Simulink: Xilinx blockset reference guide and basic tutorial*, San Jose: Xilinx Inc., 2000.