

ANALYZING FINITE TIME SERIES WITH DEDICATED NEURAL NETWORK HARDWARE

KARL MATHIA

Newport Corp.

750 National Court, Richmond, California 94804

kmathia@newport.com

ABSTRACT

The Correlation Network (CN), an efficient computational tool for the analysis of finite, real-valued time series, is introduced. When implemented on dedicated, parallel neural network hardware, this feedforward network can process signals in real time. Here the CN computes a moving-average autocorrelation algorithm in order to improve the analysis of finite time series subject to random, uniform noise. It is outlined how the CN is implemented on a commercially available neural network processor. Test results are presented.

INTRODUCTION

Neural networks have been applied and tested for many signal processing applications [1][2]. The Correlation Network (CN) for example, when implemented on dedicated neural network hardware, is an efficient computational tool for the real-time analysis of finite time-series. Here the CN improves the signal-to-noise ratio of a finite time-series by computing a moving-average autocorrelation (MAAC) algorithm. Both autocorrelation and moving average algorithms are well-known numerical tools frequently used for signal processing, statistical analysis, and other scientific and economic purposes [9].

It is further shown that the graphical representation of the average autocorrelation algorithm resembles the structure of feedforward neural network with linear transfer functions. This intuitive notion is formalized mathematically and demonstrated using numerical results. The CN architecture is derived and implemented on a commercially available neural network processor (NNP), an example of dedicated neural network hardware with parallel processing power. The NNP's performance was compared to that of generalpurpose computers in [8], demonstrating the capability for real-time signal processing. Put the introduction text here.

THE ALGORITHM

Autocorrelation sequence and moving average algorithms are common numerical tools. The autocorrelation of a random sequence is a measure for the time variation of a random signal. The moving average may be considered a noise filter [9]. Here these tools are applied to finite, real-valued time series.

Autocorrelation. The autocorrelation of an infinite time series, i.e. a continuous stream of sampled data, can be estimated from a finite time series $x_{fin}[n]$, $-\infty < n < \infty$. The sequence $x[n]$ for computing the autocorrelation is 'extracted' from $x_{fin}[n]$ via multiplication with a time window $w[n]$ of size N :

$$w[n] = \begin{cases} 1, & \text{if } 0 \leq n \leq N-1, \\ 0, & \text{otherwise} \end{cases}, \quad (1)$$

$$x[n] = x_{fin}[n]w[n] \quad (2)$$

$$= \begin{cases} x_{fin}[n], & \text{for } 0 \leq n \leq N-1 \\ 0, & \text{otherwise} \end{cases}. \quad (3)$$

This is illustrated in Figure 1.

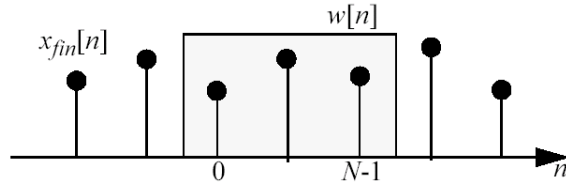


Figure 1. The time window $w[n]$ selects $x[n]$.

The autocorrelation sequence for $x[n]$, given its symmetry $y[m]=y[-m]$, is defined by

$$y[m] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n+m], \quad (4)$$

with $-(N-1) \leq m \leq N-1$. See also [9], p. 743. The autocorrelation algorithm in Equation 4 is illustrated in Figure 2. The sequence $x[n]$ is ‘fixed’ in time, while its copy $x[n+m]$ moves along the discrete time axis, beginning with the first sample at time instant $m=N-1$. For each time instant m the overlapping samples are multiplied and the products accumulated.

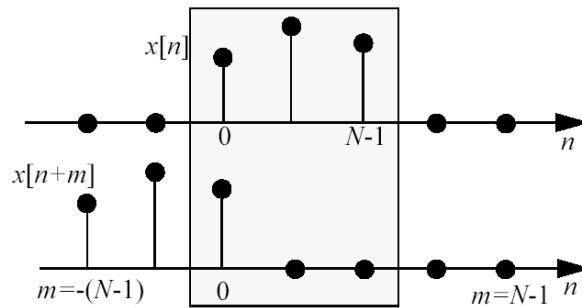


Figure 2. Illustration of the autocorrelation with $N=3$.

Moving Average. Noise in $x[n]$ also affects the autocorrelation sequence $y[m]$, but can be reduced, for example, by computing the moving average of $y[m]$ ([9], p. 840):

$$\bar{y} = \frac{1}{2N-1} \sum_{m=-(N-1)}^{N-1} y[m] . \quad (5)$$

The MAAC, the moving average of the k -th sequence $y[m]$, denoted \bar{y}_k , is determined by moving $w[n]$ and computing \bar{y}_k for the selected $x_k[n]$. See below.

THE CORRELATION NETWORK

Network Design and *A Priori* Knowledge. It is good practice to maximize the use of *a priori* knowledge for neural network design [6]. The parameters of the above algorithm provide the *a priori* knowledge for fully pre-specifying the CN. No weight training or other parameter adjustments are required. The four pre-specified parameters defining the CN are listed in Table 1. See also [5].

Table 1: Pre-specified network parameters.

Network Parameter	Pre-Specified By...
Directed Connectivity Graph	Multiply-accumulate pattern in Figure 3
Number of neurons/layer	Size N of window $w[n]$
Neuron transfer function	Product $x[n] x[n+m]$: linear operator
Connection weights w : 1	Product $x[n] x[n+m]$: no other factors

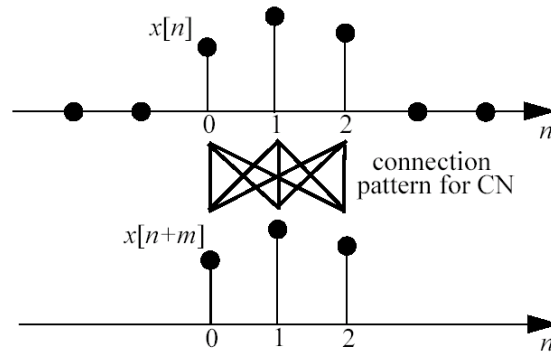


Figure 3. Obtaining the CN connection pattern, $N=3$.

The ‘connection pattern’ obtained from multiplying the sequences $x[n]$ and $x[n+m]$, is illustrated in Figure 3. Together with the network parameters in Table 1 it results in the CN architecture illustrated in Figure 4. The CN employs an $N-N-1$ structure, i.e. one hidden layer, and the number of input nodes equals the number of hidden neurons, determined by the window size N (here: $N=3$). All neurons perform the usual multiplyaccumulate operations with a subsequent

linear transfer function. Note the double connection from i -th input node to i -th hidden neuron. The CN architecture is scalable and suitable for implementation on dedicated neural network hardware, here the NNP.

Proof. It is easy to show that the CN in Figure 4 computes the average \bar{y} in Equation 5, i.e. $y_{CN} = \bar{y}$:

$$y_{CN} = \frac{1}{2N-1} \sum_{n=0}^{N-1} y[n] \quad (6)$$

$$= \frac{1}{2N-1} \sum_{n=0}^{N-1} \left(\frac{1}{N} \sum_{l=0}^{N-1} x[n]x[l] \right), l = n + m \quad (7)$$

$$= \frac{1}{2N-1} \sum_{n=0}^{N-1} \left(\frac{1}{N} \sum_{m=-(N-1)}^{N-1} x[n]x[n+m] \right) \quad (8)$$

$$= \frac{1}{2N-1} \sum_{m=-(N-1)}^{N-1} \left(\frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n+m] \right) \quad (9)$$

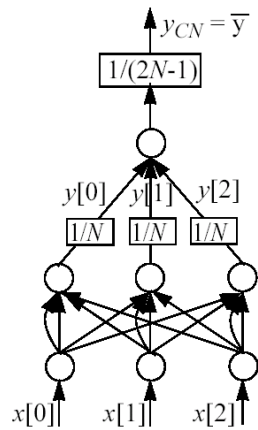


Figure 4 (left). Correlation Network for $N=3$.

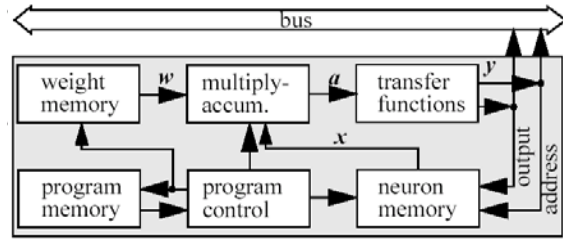


Figure 5 (right). Block diagram of a single NNP.

IMPLEMENTATION ON NEURAL NETWORKS HARDWARE

The Neural Network Processor®. The CN was tested with the commercially available Neural Network Processor NNP® by Accurate Automation Corp. [8][10]. Below its design concept is reviewed.

Architecture. The NNP® architecture provides features for fast processing of neural connections and transfer functions, and it is well suited for feedforward ANN architectures. Up to eight NNPs can be integrated in a parallel multiprocessor environment, resulting in a multiple instruction/multiple data (MIMD) platform. Figure 5 shows the block diagram of a single NNP®.

A multiply-accumulate unit, lookup tables for transfer functions, and 16-bit integer arithmetic emphasize the above feature. Figure 5 illustrates the dataflow of activation values, neuron inputs and weight values. The neuron output, i.e. the transfer function, is read from a preloaded lookup table. Weights are usually learned offline, but on-chip learning is possible [7].

Efficiency is further increased by instruction pipelining, enabling the completion of one instruction, but several data, per clock cycle (single instruction/multiple data, or SIMD). Pipelined events in overlapped time periods are often referred to as temporal parallelism, optimizing the use of a single processor. Parallelism through the replication of physical devices is referred to as spatial parallelism [3]. In this sense a single NNP with instruction pipelining implements temporal parallelism. Spatial parallelism is realized by parallel NNPs, whose performance increases linearly with the number of NNPs.

Programming Paradigm. The NNP programming paradigm emphasizes layered, well-structured, feedforward ANN architectures. Accordingly, its programming language comprises only nine powerful commands, tailored for the efficient computations of appropriate ANN primitives, in particular connections and nonlinear transfer function. For example 'mula' (multiply and accumulate) and 'lbtf' (compute transfer function). A sigmoidal neuron with three inputs and weights would be implemented using four commands:

Typical neural network	Correlation Network
mull x1, w1	mull x1, 1
mula x2, w2	mula x2, 1
mula x2, w2	mula x2, 1
lbtf c, sigmoid	lbtf c, linear

The cornerstone of an NNP assembler program is the multiply-accumulate command, which multiplies one weight value with one input value and adds the product to the existing activation value. For larger yet wellstructured neural networks a code generator in the Cprogramming language creates the NNP programs.

CN Implementation. For the Correlation Network with unit weights and linear transfer functions the above command sequence for one neuron changes (listed above). The string 'linear' points to the appropriate transfer function look-up table. NNP code equivalent to the above four lines are programmed per neuron. As such, for the CN in Figure 4 with four neurons a total of 16 lines of code, plus the program overhead, is required.

Implementation Example. Implementing the moving average autocorrelation (MAAC) on the NNP is illustrated with a simple example. Consider an infinite discrete-time sine wave with amplitude 0.5 plus uniform random noise with amplitude in [-0.5,+0.5]. This infinite signal can be represented by a finite time series $x_{fn}[n]$, $n=1..N$. Here x_{fn} is sampled with 100

Hz for 10 seconds, resulting in $N=1000$ samples (Figure 6). Its autocorrelation $y_{fin}[n]$, $n=1, \dots, 2N-1$ is an estimate for the autocorrelation of the original infinite time series. It was computed with Matlab [4] and all 1999 data points are shown in Figure 7.

How does $x_{fin}[n]$ relate to the MAAC? Assume that $x_{fin}[n]$ is measured and buffered, and five new samples become available every 0.05 seconds. As such, $x[n]$ moves along $x_{fin}[n]$ with a step size of 5, resulting in 196 sequences $x_k[n] = x_{fin}[5k+1, \dots, 5k+20]$; $k=0, \dots, 196$. The NNP was programmed to compute the MAAC for all $x_k[n]$ per Equations (4) and (5). Figure 8 shows $y_1[n]$, the first autocorrelation sequence for $k=1$, comprising $2N-1=39$ samples.

Finally, the MAAC, the average \bar{y}_k for each of the 196 autocorrelation sequences $y_k[n]$, is shown in Figure 9. The MAAC exhibits features similar to y_{fin} , i.e. oscillation frequency and amplitude. It may be concluded that the MAAC provides the information similar to the estimate of an autocorrelation for finite time series, and it can be efficiently computed on neural hardware.

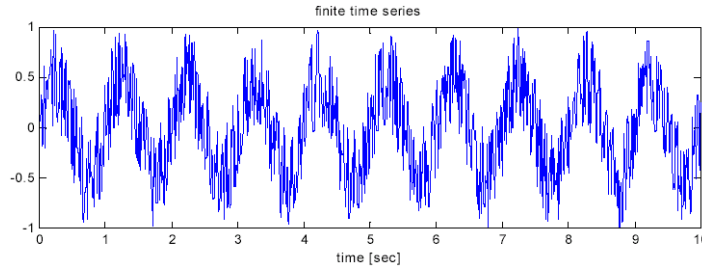


Figure 6. Finite time series $x_{fin}[n]$, $N=1000$ samples.

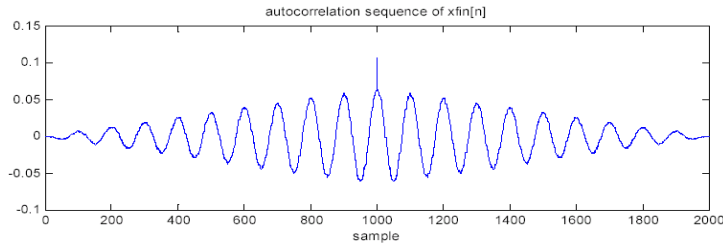


Figure 7. Autocorrelation sequence $y_{fin}[m]$ for $x_{fin}[n]$.

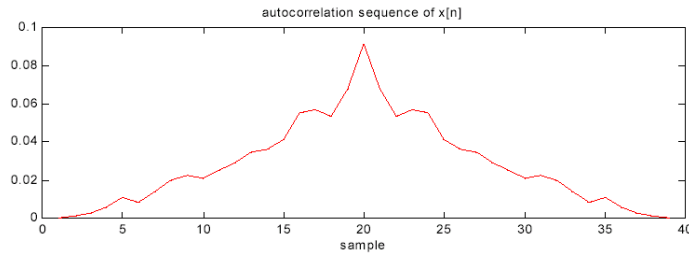


Figure 8. Autocorrelation for sequence $x_1[n]$.

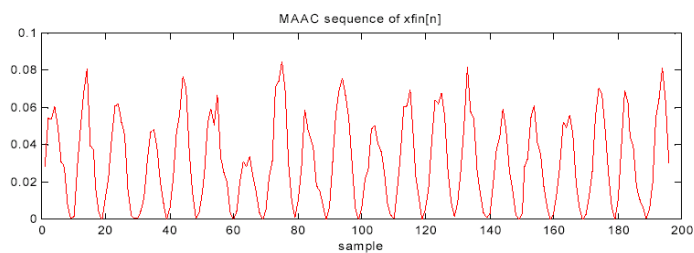


Figure 9. MAAC sequence, 196 samples.

CONCLUSIONS

The Correlation Network (CN), an efficient computational tool for the analysis of finite, real-valued time series, was introduced. When implemented on dedicated, parallel neural network hardware, this feedforward network can process signals in real time. Here the CN computes a moving-average autocorrelation algorithm in order to pre-process and improve the analysis of a finite time series subject to random, uniform noise. It was shown how the CN is implemented on a commercially available neural network processor, whose performance allows the real-time execution of the MAAC for many time-critical applications.

REFERENCES

- [1] F.L. Luo, *Applied Neural Networks for Signal Processing*, Cambridge Univ. Press, Cambridge, Mass., 1999.
- [2] S. Haykin, *Neural Networks - A Comprehensive Foundation*, IEEE Press/Macmillan College, New York, 1994.
- [3] Hennesy J.L. and D.A. Patterson, *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufman, San Mateo, California, 1993.
- [4] J. Little and L. Shure, *Signal Processing Toolbox, User's Guide*, The MathWorks, Inc., South Natick, Mass., 1988.
- [5] Karl Mathia, "A Variable Structure Learning Algorithm for Multilayer Perceptrons," Proc. ANNIE'04, Rolla, Missouri, 2004.
- [6] G.G. Lendaris and K. Mathia, "Using A Priori Knowledge to Prestructure ANNs", *Australian Journal of Intelligent Information Processing Systems*, Vol. 1, No. 1, pp. 25-30, May 1994.
- [7] K. Mathia, and J. Clark, "On Neural Network Hardware and Programming Paradigms," *Proceedings International Joint Conference on Neural Networks'02*, May 12-17, Honolulu, Hawaii, 2002.
- [8] K. Mathia, J. Clark, B. Colbert, and R. Saeks, "Benchmarking an MIMD Neural Network Processor", *Proc. World Cong. Neural Networks '96*.
- [9] A.V. Oppenheim and R.W. Schaffer, *Discrete-Time Signal Processing*, Prentice-Hall, Englewood-Cliffs, New Jersey, 1989.
- [10] R. Saeks, K. Priddy, K. Schnieder, and S. Stowell, "On the Design of an MIMD Neural Network Processor," *Proc. World Cong. Neural Networks '95*, San Diego/California, pp. 590-595, June 1995.