Neural, Parallel, and Scientific Computations 20 (2012) 475-484

TIME SERIES PREDICTION AND HARDWARE OF THE EXTENDED-INPUT NEURAL NETWORK (EINN)

WAFIK A. WASSEF

4076 Donnic Drive
Burlington Ontario L7M 0A5, Canada
wassef1@gmail.com

Abstract. The Extended-Input Neural Network (EINN) is basically a binary single layer feed-forward neural network that has special connections between the original inputs and the added inputs in order to ensure the existence of the inverse of the extended-input matrix as well as to make the calculation of this inverse of any order a simple task. The nonlinear functions (digital logic AND gates) used in the Extended-Input Neural Network are connected between the original and the added or extended-inputs instead of using them as activation functions at the output nodes. Consequently the output nodes act simply as linear binary adders and accumulators. The extended-inputs have specific number and order in which they are arranged. The exact number of the required added inputs as well as the formulas used for their generation and connections with the original inputs are given. The properties of the inverse of the extended-input matrix and how to directly deduce it are given as well. An application of the Extended-Input Neural Network for the prediction of the output states that follow a given ordered time series is demonstrated by five examples. The hardware and algorithm used for the implementation of this neural network are given.

I. INTRODUCTION AND REVIEW

THE Extended-Input Neural Network [1] (EINN) was introduced and applied for the calculation of several mathematical and logic functions. It is basically a binary, single layer feed-forward neural network that has a specific number of added inputs with special connections to the original inputs using digital logic AND gates in order to ensure the existence of the inverse of the extended-input matrix. Let this binary network has N original inputs. These inputs may assume any one of 2^N binary states. The extended-input matrix X has columns that represent the states of all the input nodes (original and added) that corresponds to the sequentially increasing original input binary states starting from $[00\ldots0]^T$ to $[11\ldots1]^T$ with Least Significant Bit (LSB) increments where the top entry in each column is the LSB. Consequently the extended-input matrix is a universal matrix that includes all possible binary combinations of the original inputs. Thus this extended-input matrix and its inverse needs to be calculated only once for all possible

Received May 6, 2012

inputs and desired outputs. The output matrix *Y* is related to the extended-input matrix *X* by the relation:

$$Y = WX \tag{1}$$

where the matrix *W* represents the synaptic weights connecting the extended input and output nodes where each entry w_{ik} connects the *i th* output node to the *k th* extended input node . No activation or nonlinear function is used at the output nodes. They are simply used as summing points only. Thus the weight matrix *W* is determined by the relation:

$$W = YX^{-1} \tag{2}$$

Consequently it is necessary that the inverse of the extended-input matrix exists and at the same time we may be able to compute it easily specially for a large order extended-input matrix. A necessary, but not sufficient, condition for the *existence* of this inverse matrix is that *X* must be a square matrix. Therefore the original number of input nodes *N* must be supplemented by other added or extended-input nodes (number of rows) in order to match the number of input states 2^N (number of columns) to make *X* a square matrix. At the same time these extended inputs must not be linear combinations of the original inputs otherwise the inverse of the extended-input matrix will not exist.

To calculate the exact number of the required added input nodes I used a special case of the binomial series [1]:

$$2^N = 1 + N + \frac{N(N-1)}{2!} + \frac{N(N-1)(N-2)}{3!} + \dots \tag{3}$$

The first term on the right hand side of (3) represents a single constant input which is chosen to have a constant binary value of *1* for all input states, the second term *N* represents the number of the original inputs, the third term represents the number of all possible combinations of taking two of the original *N* inputs at a time. Each pair of these two original inputs is connected to the inputs of a 2-input AND gate. The outputs of these AND gates are added to the original inputs to become part of the extended-inputs of the neural network. The next term is the number of all possible combinations of taking three of the original *N* inputs at a time and are connected to the inputs of 3-input AND gates. The outputs of these AND gates are part of the extended-inputs to the neural network, and so on. As an example the case for *N=4* was given before [1] and is shown in Fig. 1 below. The inputs at the bottom of Fig. 1 are the original inputs and a constant input *1*. The top of this figure (the square nodes) shows the original and the added inputs together and I will refer to these nodes *together* from now on as the extended-inputs.

The order in which each extended-input *m* (starting from the most left at *m* = 1 at the top of Fig.1) is located can be determined as follows: denoting the values of the original inputs at the bottom of Fig.1 by *n* (*n* = 1 for X1, *n* = 2 for X2, *n* = 3 for X3 and so on), then the *m th* extended input that is directly connected to one of the original inputs is calculated as follows:

$$m = 2^{n-1} + 1 \tag{4}$$

Thus the original inputs are directly connected to the extended inputs no. 2, 3, 5 and 9. For those extended inputs connected to the outputs of 2-input AND gates we have:

$$m = 2^{n_1 - 1} + 2^{n_2 - 1} + 1 \tag{5}$$

where n_1 and n_2 are the values of the two original inputs connected to the inputs of each of the 2-input AND gate and assume values similar to n above. This can be extended to 3-input and 4-input AND gates and so on. In this case the extended input matrix takes an upper triangular form and its inverse is shown (see section II below) to be identical to the extended-input matrix itself except for the fact that all the non-zero entries of some of its sloping down diagonals (parallel and above the main diagonal) have values of – *1* instead of +*1*. The extended inputs of the neural network are arranged in such a way as to produce an input matrix and its inverse with similar forms. The case for N = 4 are given by (6) and (7) below.

As the states of the original inputs are incrementally and sequentially increased the extended-inputs are activated or “fired” one after the other in order starting from the most left node and extending to the right. This is a consequence of the fact that X is an upper triangular matrix.

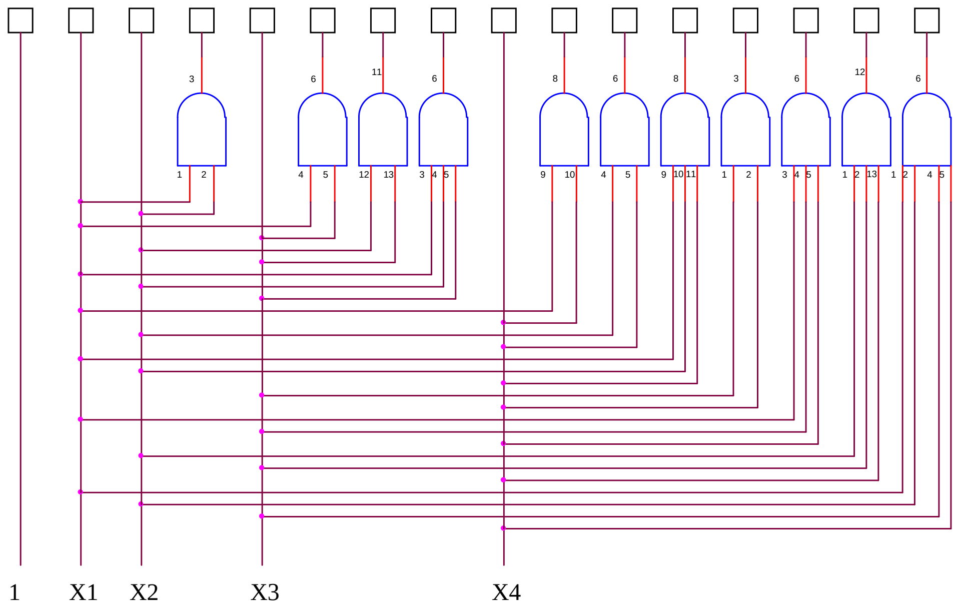

Fig. 1. The connections between the extended-input nodes (top) and the original inputs (bottom) through multi-input logic AND gates.

$$X = \begin{bmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\
0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix} \tag{6}$$

$$X^{-1} = \begin{bmatrix}
+1&-1&-1&+1&-1&+1&+1&-1&-1&+1&+1&-1&+1&-1&-1&+1\\
0&+1&0&-1&0&-1&0&+1&0&-1&0&+1&0&+1&0&-1\\
0&0&+1&-1&0&0&-1&+1&0&0&-1&+1&0&0&+1&-1\\
0&0&0&+1&0&0&0&-1&0&0&0&-1&0&0&0&+1\\
0&0&0&0&+1&-1&-1&+1&0&0&0&0&-1&+1&+1&-1\\
0&0&0&0&0&+1&0&-1&0&0&0&0&0&-1&0&+1\\
0&0&0&0&0&0&+1&-1&0&0&0&0&0&0&-1&+1\\
0&0&0&0&0&0&0&+1&0&0&0&0&0&0&0&-1\\
0&0&0&0&0&0&0&0&+1&-1&-1&+1&-1&+1&+1&-1\\
0&0&0&0&0&0&0&0&0&+1&0&-1&0&-1&0&+1\\
0&0&0&0&0&0&0&0&0&0&+1&-1&0&0&-1&+1\\
0&0&0&0&0&0&0&0&0&0&0&+1&0&0&0&-1\\
0&0&0&0&0&0&0&0&0&0&0&0&+1&-1&-1&+1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&+1&0&-1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&+1&-1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&+1
\end{bmatrix} \tag{7}$$

If these sequentially increasing states of the original inputs represent the unidirectional flow of discrete time then this Extended-Input Neural Network converts Time into a spatially ordered structure. This observation has implications on the successive and growing activation of the synaptic weights connecting the output nodes and the extended-input nodes starting first from the most left extended-input node then extending step by step to the right as the time series progresses. This remark will be used in the time series prediction given in section III below.

II. PROPERTIES OF THE EXTENDED-INPUT MATRIX

Some of the special properties of the extended-input matrix and its inverse are the following:

1. The extended-input matrix is an upper triangular matrix. Consequently its inverse is an upper triangular matrix as well [2].
2. The absolute magnitude of each entry of the inverse of the extended-input matrix equals to the corresponding entry (at the same row and column) of the extended-input matrix: each zero corresponds to a zero and each absolute magnitude of *1* corresponds to a *1*.
3. The sum of all the entries in each column, except the first column, of the inverse of the extended-input matrix equals zero.
4. All the non-zero entries of each of the sloping down diagonals (the main diagonal and all other parallel diagonals above it) of the inverse of the extended-input matrix have the same sign.
5. The values of the synaptic weights of this neural network for any binary input and output states are zero or positive or negative integers.
6. Both the extended-input matrix and its inverse are symmetrical with respect to the diagonal perpendicular to the main diagonal (extending from the lower left corner to upper right corner). For example each entry of the inverse matrix satisfies the relation:

$$z_{k,m} = z_{M+1-m,M+1-k} \tag{8}$$

where $M = 2^N$ is the order or size of the extended-input matrix and its inverse.

We may use the following procedure in order to write down the exact inverse matrix of the extended-input matrix of any order:
1. Write down the inverse matrix exactly similar to the extended-input matrix itself (property 2).
2. Since $z_{11} = +1$ then all the entries of the main diagonal of X^{-1} must remain positive and equal to $+1$ (property 4).
3. Since $z_{12}+z_{22} = 0$ (property 3) then all the non-zero entries of the first sloping down diagonal (above and parallel to main diagonal) must equal *-1*.
4. Since $z_{13}+z_{23}+z_{33} = 0$ then all the non-zero entries of the second sloping down diagonal must equal *-1*….and so on.

III. TIME SERIES PREDICTION

The sequentially increasing ordered *N*-bit binary original inputs makes this Extended-Input Neural Network fit naturally to be used for a time series prediction in which the LSB increments of the original inputs are the intervals of time at which the binary output is sampled. If we know the first ordered 2^N -1 output states then we can predict the next output state (last column of *Y*) as follow:
1. Eliminate the last (bottom) row of X^{-1} so that its number of rows matches the number of columns of *Y* (without the last column which is the output state to be predicted).
2. Apply (2) to calculate the weight matrix *W*.
3. Reset all the non-zero entries of the last column of *W* to zero.
4. Substitute this new *W* matrix into (1) (using the full *X* matrix) to find the complete output matrix *Y* which includes the last state to be predicted (last column).
5. Shift the sampled output states one time increment to the right and repeat the same procedures (for a new neural network) to predict the next output state and so on.

The time series given in the examples below do not require an extended-input matrix of order more than 16x16. However for more complex time series the first 15 output states may not be sufficient as a memory depth for the prediction of the next (future) output state. Consequently these complex time series may require an extended-input matrix X of higher order.

The multiplication of the truncated inverse of the extended-input matrix and the incomplete output matrix give the exact values of all the weights except those weights in the last column of the weight matrix W. This is a consequence of the fact that the inverse of the extended-input matrix has an upper triangular form. The errors in the weights in the last column are due to the lack of knowledge of the last (future) output state. However these weights in the last column of W are connected between the output nodes and the last extended input (most right) node which is activated only when the last original input state $[1111]^T$ is reached (see Fig. 1). This last input (future) state, by definition, has not occurred yet. Consequently the weights connected to this last extended input node should not be activated yet. The spatially ordered extended-input nodes separate between the weights according to their temporal sequence of activation. According to the principle of causality the only connections of the neural network that should be used to predict this last (future) output state are those connections that have been already established and precisely calculated during the past (given) $2^N - 1$ ordered output states. Consequently we must reset all the entries of this last column of W to zero as indicated in the above procedures.

The prediction of the output state that follows a given time series is demonstrated successfully by the following five examples (for $N = 4$, thus using (6) and (7) above):

1. The output is increasing linearly from $[0000]^T$ to $[0111]^T$ with LSB steps at every increasing increment of the original input state (Y is a 4x15matrix only). The next output state was correctly predicted to be $[1111]^T$.
2. The output is decreasing linearly from $[1111]^T$ to $[1000]^T$ with LSB steps at every increasing increment of the original input state. The next output state was correctly predicted to be $[0000]^T$.
3. The output is oscillating between $[0000]^T$and $[1111]^T$ and vice versa at every increasing increment of the original input state. The next output state was always correctly predicted to be the other state compared to its previous state.
4. The output is a triangle wave form (0, 2, 4, 6, 8, 6, 4, 2, 0, 2, 4, 6, 8, 6, 4 expressed in binary numbers). Each step of the output is at every increasing increment of the original input state. The next output state was correctly predicted to be binary 2: $[01000]^T$. This example was also tried successfully with different starting points on the same time series.
5. Each output state is the square of the corresponding original input state which increases sequentially from $[0000]^T$ to $[0111]^T$ (this time series is a discrete parabola and Y is an 8x15 matrix). The next output state was correctly predicted to be $[10000111]^T$ (binary $225=15^2$).

IV. ALGORITHM AND HARDWARE

The algorithm for computing the weight matrix W for general purpose applications without using matrix multiplication as in the last section consists of three steps:

1. Generate the extended input matrix X from the original inputs using (4), (5) and similar equations for AND gates with larger number of inputs.
2. Convert the extended input matrix into its inverse X^{-1}using the properties and the procedures given in section II above.

3. Modify each column of the inverse of the extended-input matrix by resetting some of its entries to *0* (according to the following rule) then add all the remaining non-zero entries in this column to produce the corresponding weight. The rule is as follows:

$$w_{ij} = \sum_{k=1}^{j} y_{ik} z_{kj} \tag{9}$$

The upper limit of the summation is due to the fact that the inverse of the extended-input matrix is an upper triangular matrix. This feature reduces the calculations by half. It is clear from (9) that the products of each entry y_{ik} of the output matrix Y and the corresponding entry z_{kj} of X^{-1} will leave it unchanged if $y_{ik} = 1$ or reset it to zero if $y_{ik} = 0$. After this step we add up all the remaining non-zero entries in this column of the modified inverse matrix to determine the corresponding weight w_{ij}. If this row of Y (which represents all the ordered states of the i th output node) has all its entries equal to *1* then in this case we use property 3 in section II above for the inverse matrix to realize that there is no weight connected to this output node from any extended input node except from the first constant input node *1*. Thus this neural network automatically eliminates any connections (except from a constant source) that produce unchanging outputs in response to the changing inputs. After we obtain all the entries of W in this way we save this weight matrix to be recalled and used with the hardware given below.

The properties of the extended-input matrix stated above make the implementation in hardware of the present binary neural network much simpler than that used for a fully connected feed forward neural network. Some of the features that simplify this implementation are the following:

1. All the weights are positive or negative integers hence we may use a shift register to produce these weights.
2. The negative integer weights can be implemented easily for binary numbers by taking the twos complement of the positive integers using simple and available digital circuits [3].
3. The output nodes are summing points only without using any activation (nonlinear) functions. Thus we may use digital adders or accumulators at the output nodes.
4. All the weights are obtained by modifying the standard and universal inverse of the extended-input matrix according to the desired output. Consequently we do not have to choose randomly the initial values of these weights and then apply time consuming iteration and optimization techniques.
5. The available advanced fast digital circuits suggests that the massively parallel connections between output and extended input nodes may be replaced by a single circuit and then multiplex the extended inputs to this circuit and de-multiplex the output of this circuit to the output nodes.

Figure 2 below shows the hardware that may be used to implement the Extended-Input Neural Network using this single circuit. The extended inputs may be generated either by hardware (AND gates) similar to the circuit shown in Fig. 1 above or by software which employs (4) and (5) and similar equations to generate those extended inputs and then store them in a look-up table. As the number of the original inputs N increases, the number of the extended inputs increases exponentially (2^N). Consequently it may be easier to produce the extended inputs using software. The hardware is shown in Fig. 2 below.

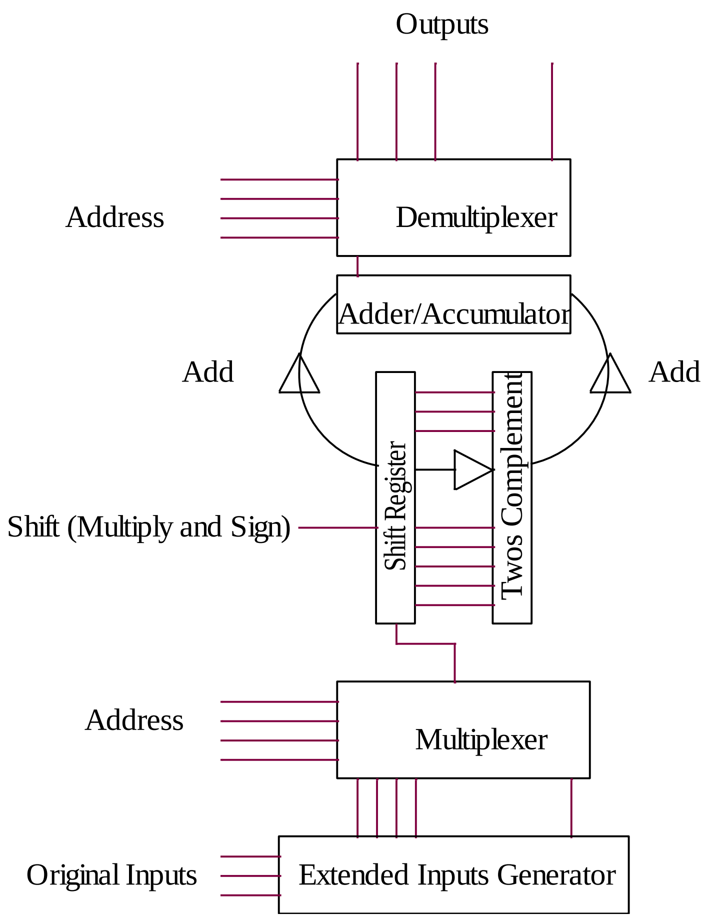

Fig. 2. The hardware used for the implementation of the Extended-Input Neural Network for general purpose applications.

First we select one of the output nodes by the address of the de-multiplexer (or decoder). For each output node the address of the multiplexer steps through all the extended-input nodes which are connected to this particular output node (only when a non-zero weight is connected between them) one input node at a time. A fact that reduces further the number of the extended inputs that contribute to this particular output is that we need only to consider the extended inputs at a high state *1*. The next step is to load this bit *1* of the extended input into the least significant bit (LSB) of the shift register. The shift register is then left-shifted a number of bits depending on the value of the weight (one left shift = 2, LSB + one left shift =3, two left shifts = 4 and so on). If the weight is negative, then the twos complement of the shifted binary number is produced. An adder/Accumulator is used to sum up all the contributions from all the weighted extended-inputs connected to this particular output node. The output of the adder/Accumulator is loaded into the de-multiplexer. Then the address of the de-multiplexer is incremented to the next output node and the whole process is repeated for the rest of the output nodes.

In this way we do not have to duplicate this hardware for each weight connecting every extended-input node and every output node similar to the fully connected feed-forward neural network. Instead we multiplex these nodes one at a time using only a single circuit.

Thus we may use this hardware for any large number of input and output nodes. The parallelism of the neural network is replaced by multiplexing the extended inputs to the available extremely fast digital circuits.

V. CONCLUSION

The lateral extension of the Extended-Input Neural network resembles the extension of the surface area of the cerebral cortex by its corrugated shape, and the multiple connections of these added inputs to the original inputs through multi-input logic AND gates mimic the many dendrite branches feeding into each neuron. This structure produced an extended-input matrix and its inverse with similar forms, a feature that greatly facilitated the calculation of the exact inverse of the extended-input matrix of any high order. The upper triangular form of the extended-input matrix converted the sequentially increasing input states which resembles the unidirectional flow of discrete time into a spatially ordered structure of the extended-input nodes which are activated sequentially and in order. Consequently the upper triangular forms of these matrices enabled us to accurately calculate the synaptic weights connecting the output and the extended-input nodes. The errors introduced in the values of the synaptic weights connected to the last extended-input node are due to the lack of knowledge of the future state to be predicted. These weights are activated only when this last (future) state occurs. Consequently they are reset to zero values. It is demonstrated by five examples how the Extended-Input Neural Network is capable of accurately predicting the output states that follow given time series of diverse mathematical forms. An algorithm and hardware are given for the computation of the synaptic weights and the implementation of this neural network for general purpose applications. In these computations we modify the standard and universal inverse of the extended-input matrix according to the desired output. The hardware described here deals with the problem of the exponentially increasing number of the extended-inputs with the increase of the number of the original inputs by multiplexing these extended input and output nodes one node at a time and thus using only a single circuit made up of a shift register, twos complement digital circuit and a binary adder/accumulator. This hardware is therefore much simpler than that used in some of the other neural networks.

REFERENCES

[1] W. A. Wassef, "Extended-input neural network: application, implementation and learning," Journal of Neural, Parallel and Scientific Computation, vol. 10 no. 4, Dynamic Publisher, Atlanta GA, pp.387-410, 2002.

[2] H. Anton, *Elementary Linear Algebra*. 8th ed. New York: John Wiley & Sons, 2000, p. 68.

[3] W. Stallings, *Computer Organization and Architecture - Designing for Performance.* 7th ed. NJ Pearson Prentice Hall, 2006, pp. 296-301.