Transactions on Information and Communications Technologies vol 20, © 1998 WIT Press, www.witpress.com, ISSN 1743-3517

A Fuzzy Recurrent Neural Network of Binary Neurons for Content Addressable Memory

Roelof K Brouwer
University College of the Cariboo
Kamloops, BC
V2C 5N3
CANADA
email rbrouwer@cariboo.bc.ca

Abstract

This paper is concerned with a proposal for a recurrent neural network of fuzzy neurons which may be used as a content addressable memory. The behavior of the fuzzy unit in the network is based on fuzzy logic in that each component of the binary input vector to the fuzzy neuron is compared to a number which represents the membership value for a 0 in that position. The results of the comparisons are then combined using a generalized mean function to produce a single number which is compared to a threshold. A training algorithm is developed based on an algorithm for linear inequalities described by Ho and Kashyap in a paper titled " An Algorithm for Linear Inequalities and its Applications". The results obtained by simulation of this content addressable memory look promising enough to warrant further investigation.

1. Introduction

McCulloch and Pitts [1] proposed a neural network model of distributed memory in 1943. Amari [2,3] was the first to propose a rigorous mathematical foundation for the neural network model while Hopfield[4] generated new interest in neural networks by proposing their use as a content addressable memory. A Hopfield network is a recurrent network

which is fully connected with no self loops and symmetric synaptic weights. Learning for a Hopfield network was prescribed by use of the Hebbian[5] rule. Hebbian learning is quite restrictive because it results in relatively small storage capacity. Even though patterns may be potentially learnable, Hebbian training may not allow the storage potential to be reached and may fail in that the required weights may not be calculated.

This paper is concerned with a proposal for a fully recurrent Hopfield style artificial neural network with fuzzy neurons. We consider a network of two-state units with each state represented by 0 and 1. The purpose of the learning algorithm is to assist in finding a set of parameters that allows given patterns to be stored in the network as fixed points or stable states.

The network is similar to a Hopfield network because it is fully recurrent although symmetry is not required and self connections are allowed. The fuzzy neurons in the network proposed here also behave differently from the crisp or non-fuzzified neurons which make up the Hopfield network. The fuzzy neuron is based on fuzzy logic in that each component of the binary input vector to a neuron is compared to a number which represents the membership value for a 0 in that position. The results of the comparisons are then combined using a generalized mean function to produce a single number which is compared to a threshold. In the case of a non-fuzzy or crisp perceptron or neuron, which is used in the Hopfield network, a linear combiner with hard limiting function[6] is used to generate the single output of a neuron.

A training algorithm based on an algorithm for linear inequalities described by Ho and Kashyap [7,8] may be used to determine the fuzzy sets for a unit. The results obtained by simulation of the network for storing memories look promising.

The paper commences with a brief introduction to fuzzy logic and fuzzy sets. This is followed by a brief explanation of a perceptron and the transform function for a fuzzy neuron. A training algorithm for the proposed perceptron and thereby the complete network is developed next. Results of simulations are used to demonstrate the effectiveness of the training algorithm for storing memories.

2. Fuzzy Sets

The applications of fuzzy logic and neural networks are related as follows. While both fuzzy logic and fuzzy sets are useful for representing, manipulating, and using inexact information as it exists in our brains, neural networks provide computational power, learning capabilities and

robustness. A seminal work on neural networks which revived interest in neural networks is a paper by Hopfield [5]. Fuzzy sets are a means for expressing the vagueness of inexact information. These sets can be thought of as a generalization of classical set theory and were introduced by Zadeh in 1965 in his seminal works [9,10] as a mathematical way to represent vagueness in linguistics.

Fuzziness is the vagueness found in the definition of a concept or the meaning of a term such as ‘’tall person” or “large ship”. Fuzziness implies non-distinctiveness. Elements may belong to a fuzzy set to various degrees. For example the set of heights of tall men is a fuzzy set. A height of 6' belongs to this fuzzy set to a lesser degree than a height of 6'3". The reason for the fuzziness is that the concept of tall is fuzzy. The opposite of a fuzzy set is a crisp set and the characteristic function for crisp sets is replaced by membership functions for fuzzy sets. The range of the characteristic function for a crisp set is {0,1} while the range for a membership function for a fuzzy set is [0,1].

Fuzzy logic is a methodology for dealing with fuzzy sets. Let X be a universal set, a typical element of which is represented by x. Let A be a fuzzy set with membership function $\mu_A(x)$ which has values restricted to [0,1]. $\mu_A(x)$ then expresses the degree to which x is compatible with the concept expressed by A. The intersection , union and complement of fuzzy sets is defined by the expressions $\mu_{A\cup B}(x)$= max ($\mu_A(x)$,$\mu_B(x)$), $\mu_{A\cap B}(x)$ = min($\mu_A(x)$ $\mu_B(x)$), and $\mu_{A'}(x)$ = 1 - $\mu_A(x)$.

3. Perceptrons

A perceptron, shown in Figure 1 below, may be regarded as a linear combiner followed by a hard limiter[6]. It is described by the transform

$$\mathrm{sgn}(\mathbf{w}.\mathbf{x}-t) \qquad (1)$$

x is the input vector, **w** is the weight vector, t is the threshold and sgn(s) ≡ (s ≥ 0) - (s<0).

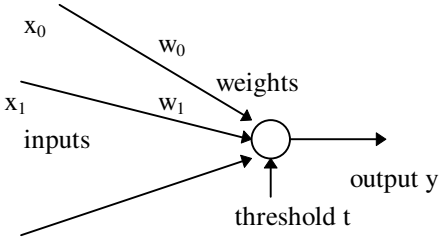

Figure 1. perceptron

The latter is called the threshold function. Note that the convention that is used here is that false is 0 and true is 1. The mapping function in equation 1 corresponds to a hyperplane which separates input vectors into two classes [11]. The value of the weight vector may be obtained by the perceptron learning algorithm, also called Rosenblatt learning algorithm [12], or by the delta rule [13]. Note that bolded numbers and letters are generally vectors in this document.

4. Fuzzification of a Neuron

There are basically two kinds of fuzzy neural models [14,15]. One is a model whose input-output relations of neurons are described by fuzzy IF-THEN rules while the other comes from fuzzification of nonfuzzy neuron models. An example of the latter is the fuzzy perceptron proposed by Keller and Hunt [16]. In their proposal the perceptron training algorithm is simply fuzzified. Other types of fuzzy neural models are discussed in [14]. One, called a fuzzy neuron of type I is a fuzzy neuron with crisp signals and fuzzy weights. The other is a fuzzy neuron, called a fuzzy neuron of type II, with fuzzy signals which are combined with fuzzy weights. A fuzzy neuron of type III is a fuzzy neuron described by fuzzy logic equations. The fuzzy neuron described in this paper is of type I. The weights, w_i are replaced by membership functions μ_{Ai}. A single weighted input, $x_i * w_i$ is replaced by a membership value $\mu_{Ai}(x_i)$. Membership values are aggregated to produce a single value in [0,1]. The resulting value may then be compared to a threshold value as shown in Figure 2 below.

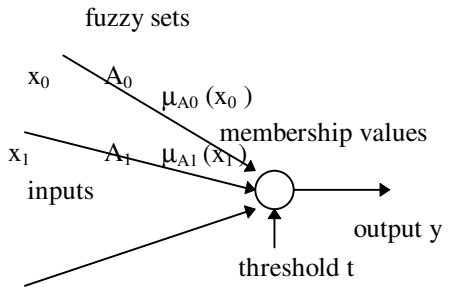

Figure 2. fuzzy binary neuron

5. Fuzzy Binary Input Neuron

For the fuzzy binary input neuron described here the inputs are restricted to binary input such that $x_i \in \{0,1\}$ and $\mathbf{x} \in \{0,1\}^n$. Each of the input lines to the neuron corresponds to a fuzzy set, denoted by A_i, rather than a single weight. Each of the fuzzy sets has only two members with non-zero membership values. The function of an input line to the neuron is to determine the membership value of the input value on the input line. Let the membership value for the value 0 on the i^{th} input line be denoted by m_i. The membership value for 1 on the i^{th} input line is set to the complement of 1 - m_i. Thus each fuzzy set A_i is completely defined by one value m_i since input is restricted to be binary. These membership values generated by the input lines must be combined some way and there are various ways for doing this [14]. One such way is the generalized mean, which for a vector of values (v_0, v_1, …… v_{n-1}) is defined as

$$\left(\frac{\sum_{i=0}^{n-1} v_i^r}{n} \right)^{i/r} \tag{2}$$

In this paper a value of r = 2 will be used in expression 2. Since the output of input line *i* is m_i for x =0 and 1- m_i for x=1, the output, v_i ,can be expressed as

$$v_i =. |m_i - x_i| \tag{3}$$

Since the aggregation function for r = 2 is

$$L = \sqrt{\frac{\sum_{i=0}^{n-1} v_i^2}{n}} \tag{4}$$

$$L = \|\mathbf{m}-\mathbf{x}\| / \sqrt{n} . \tag{5}$$

This is then the output of the neuron prior to thresholding. L will be large if **m** and **x** differ which occurs if the components of the input have high membership values. Since $m_i \in [0,1]$ and $x_i \in \{0,1\}$ the maximum value of L is 1 and the minimum value is 0. Therefore a threshold, t, for the quantized output,

$$y_q = L \geq t \tag{6}$$

would have to be an element of (0,1).

6. Training Algorithm for the Fuzzy Neuron

A training algorithm for obtaining the membership values will now be considered. Let G = L - t and let d_q be the desired output for input vector **x**. The objective is that

$$G(\mathbf{x}) \geq 0 \text{ for } d_q = 1 \quad \text{and} \quad G(\mathbf{x}) < 0 \text{ for } d_q = -1 \tag{7}$$

A sufficient condition is that

$$(G(\mathbf{x}))\ (2d_q-1) > 0 \tag{8}$$

This may be changed to an equality as shown below [7]

$$(G(\mathbf{x}))(2d_q-1) = b \qquad b > 0 \tag{10}$$

or

$$J = G(\mathbf{x})(2d_q - 1) - b = 0 \qquad b > 0 \tag{11}$$

This is equivalent to

$$J^2 = 0 \qquad b > 0 \tag{12}$$

J^2 is always non-negative so that a solution to 12 would be a minimum. Since it is differentiable the gradient may be used to find a minimum. Taking the gradients

$$\nabla J^2 = 2J\nabla J \tag{13}$$

$$\nabla_{\mathbf{m}} J = (2d_q -1)(\mathbf{m}-\mathbf{x})/ L(\mathbf{m}-\mathbf{x}) \tag{14}$$

$$\nabla_t J = -(2d_q -1) \tag{15}$$

$$\nabla_b J = -1 \tag{16}$$

Equation 8 must be true for all input vectors $\mathbf{x}^{(k)}$ and corresponding desired outputs $d_q^{(k)}$ yielding *p* equations each with its own value of b i.e. $b^{(k)}$

Therefore the complete set of equations which must be used iteratively is

$$L(\mathbf{m}\text{-}\mathbf{x}^{(k)}) = \|\mathbf{m} - \mathbf{x}^{(k)}\| / \sqrt{n} \tag{17}$$

$$J^{(k)} = (L(\mathbf{m}\text{-}\mathbf{x}^{(k)}) - t)(2\, d_q^{(k)} - 1) - b^{(k)} = 0 \tag{18}$$

$$\Delta\, \mathbf{m} = -2\mu_1 J^{(k)}\, (2d_q^{(k)} - 1)(\mathbf{m}\text{-}\mathbf{x}^{(k)}) / L(\mathbf{m}\text{-}\mathbf{x}^{(k)}) \tag{19}$$

$$\Delta\, t = 2\mu_2 J^{(k)}(2d_q^{(k)} - 1) \tag{20}$$

$$\Delta b^{(k)} = 2\mu_3 J^{(k)} \tag{21}$$

Equations 17 to 19 are calculated with k = 0, 1 p-1 in sequence or in random order making up one epoch which is one learning iteration. During an epoch, each training pattern is presented once. $G(\mathbf{x}^{(k)})$ and $J^{(k)}$ are updated after 19 and 20. Note that $J^{(k)}=0$ k=0,1,--p-1 is a sufficient condition but not a necessary condition for the desired mapping to be incorporated. Values for the components of **m** and t are initially set at 0.5. Small positive values are selected for $b^{(k)}$ k=0,1,--l-1. $b^{(k)}$ is never allowed to become non-positive. If $\Delta b^{(k)}$ is such that the new b_k is ≤ 0 $\Delta b^{(k)}$ is set to 0. **m** must always be $\in [0,1]^n$. and t must always be $\in (0,1)$.

7. Training Algorithm for the Network of Neurons

In case of synchronous updating, with one state change, the recurrent network behaves as a one layer feedforward network. Below, in figure 3, is a drawing of a two-layer feedforward network to show that after two transitions or state changes the output of a neuron will be dependent on all neurons.

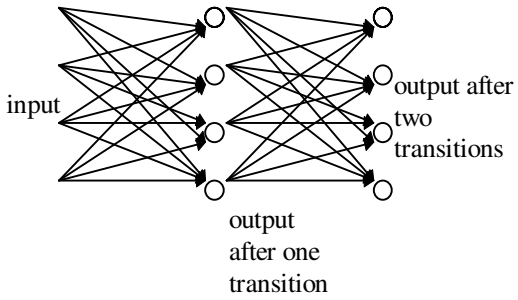

Figure 3. Two layer feed forward network

In case of training for fixed points the output after one transition, and therefore after any number of transitions, must be identical to the input. The network may be trained to store content addressable memories by training each neuron separately. This is possible since the dynamics of the network in case of synchronous updating are such that all units change state simultaneously during a single change in state. The output of any neuron in this case depends only on the input vector and the membership vector for the neuron. For the auto-associative memory considered here, neuron i must have the value of the i th component of the input vector as its output. Thus neuron 0 is trained on all desired memories, then neuron 1 is trained on all desired memories. If parallel processing were possible all neurons could be trained simultaneously. After each neuron has been trained independently it is determined how many of the desired memories map correctly to themselves using all the neurons.

8. Simulations

Hassoun [8] mentions six important characteristics of a dynamic associative memory of which the preceding network is an example. These characteristics, which pertain to retrieval dynamics following training, are (1) High capacity for stored memories (2) Convergence within a small number of transitions (3) Existence of relatively few unintended fixed states or spurious memories (4) Tolerance to partially correct inputs (5) Provision for a no decision or ground state which attracts complete undesirable input states (6) Autoassociative and/or heteroassociative processing capabilities. Since the network is a result of training, these characteristics may also be attributed to the training itself. A strictly training characteristic is the learning speed.

Simulations, to determine the extent to which the characteristics are satisfied, were carried out with various values for the number of fixed

points and the number of neurons. Following are the results of the simulations.

Training is carried out a neuron at a time. Learning speed is measured in terms of epochs. At the end of each epoch for a particular neuron the percent correct is calculated and if it exceeds the best so far the membership matrix corresponding to the best so far is replaced. This is part of the pocket algorithm described in [17]. The values for the learning rates, μ_1 , μ_2 and μ_3 were set to 0.001. The retrieval results below in each row are based on 400 randomly generated input patterns.

Table 1 Results of Simulations

			training time		retrieval results	
# funda mental	# units or neur.	load factor	average number of epochs for a unit	# states conver. to fund. mem.	# states conver. to spur. stat.	# spur. states
10	10	1	2	23	377	153
15	15	1	2	6	396	263
20	20	1	3	1	399	376
25	25	1	3	0	400	400
20	25	0.8	3	2	398	390
15	25	0.6	2	0	400	332
10	25	0.4	2	4	396	171
5	25	0.2	1	3	48	397
3	25	0.1	1	89	7	311
3	20	0.15	1	122	8	278

The load factor is the number of memories divided by the number of neurons. Spurious states are attractors which are not intended to be attractors unlike fundamental states or memories. As expected as loading is increased the number of random states which converge to non-fundamental states increases.

9. Summary

A transform function and a training algorithm for a fuzzy neuron of type I with binary input to be used in a fully recurrent network has been

demonstrated. Training of the recurrent network is done by training individual neurons. The training algorithm for an individual neuron is based on an algorithm for linear inequalities developed by Ho and Kashyap. This algorithm is also used in HoKashyap recording [2]. Simulations have demonstrated that the proposed transform is effective as a discriminant function and that the proposed training algorithm can be used to store fixed points in a fully recurrent network.

Acknowledgments

This work was supported by a Scholarly Activity Grant supplied by the University College of the Cariboo.

References

[1]. McCulloch, W.S. and Pitts, W., "A Logical Calculus of the Ideas Imminent in Nervous Activity," Bulletin of Mathematical Biophysics, Vol. 5, 1943, pp. 461-482.
[2]. Amari, S., Yoshida, K., and Kanatani, K., "A Mathematical Foundation for Statistical Neurodynamics," SIAM Journal of Applied Math., Vol. 33, No. 1, 1977, pp. 95-126.
[3]. Amari, S., "On Mathematical Methods in the Theory of Neural Networks," International Conference on Neural Networks, Vol. 3, 1987, pp. 3-10.
[4]. Hopfield, J. J. “Neural Networks and physical systems with emergent collective computational abilities” Proc. Nat. Acad. Sci. USA 79:pp. 2554-2558,1982
[5]. Hebb, D.O., The Organization of Behavior, John Wiley and Sons, New York, 1949.
[6]. Shynk, J. and Roy, S. Analysis of a Perceptron Learning Algorithm with Momentum Updating. ICASSP 90 1990 International Conference on Acoustics, Speech, and Signal Processing April 3-6
[7]. Ho, Y. C. and Kashyap, R. L. “ An Algorithm for Linear Inequalities and its Applications” *IEEE Transactions on Electronic Computers* Vol. EC-14, No. 5 October, 1965 pp. 683-688

[8]. Hassoun, M. H. Adaptive Ho-Kashyap Rules for Perceptron Training. IEEE Transactions on Neural Networks, Vol. 3, No. 1, pp. 453-465, January 1992

[9]. Zadeh, L. A. "Outline of a new approach to the analysis of complex systems and decision processes." IEEE Transactions on Systems, Man, and Cybernetics SMC-3, pp. 28-44, 1973

[10]. Zadeh, L. A. "Fuzzy Sets" Information and Control 8:pp. 338-353, 1965

[11]. Nilsson, N. J. Learning Machines McGraw Hill, New York, N. J. 1990

[12]. Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", Psych. Rev. 65: pp. 386 - 408, 1958

[13]. McClelland, T. L. and Rumelhart, D. E. , and the PDP Research Group, 1986 Parallel Distributed Processing. Cambridge: The MIT Press.1986

[14]. Lin, C. and Lee, C. S. Neural Fuzzy Systems Prentice Hall Saddle River NJ., 1966

[15]. Gupta, M. M. and Qi, J. "On fuzzy neuron models." Proc. Int. Joint Conf. Neural Networks, Vol. II, pp. 431-436, Seattle,WA 1991

[16]. Keller, M. and Hunt, D. J. "Incorporating fuzzy membership functions into the perceptron algorithm." IEEE Trans. Pattern Anal. Mach. Intell. PAMI-7(6): pp. 693-699, 1985

[17]. Gallant, S. I. "Three Constructive Algorithms for Network Learning". Cognitive Science Society Eighth Annual Conference Proceedings, Aug. 15-16, pp. 652-659. 1986