Pattern Classification Using Teurons

M. Sipper and Y. Yeshurun

Department of Computer Science, School of Mathematical Sciences
Sackler Faculty of Exact Sciences, TEL AVIV UNIVERSITY

Abstract

Neural Networks consist of simple elements, capable of summation and thresholding. We define a more general element - the "Task Oriented Neuron" or Teuron, that can compute higher order functions. We further define Teuron Networks, and show two such networks that may be used as content addressable memories and as pattern classifiers.

The first network is based on the Gödel encoding scheme. It uses this scheme in order to memorize sequences of numbers. These sequences are then stored analogically. The second network uses binary encoding , i.e. a binary sequence is translated into its decimal equivalent and then stored analogically.

We shall motivate the use of such networks by discussing the feasability of their implementation. Our conclusion is that they may be implemented in such a way so as to reduce cost due to reduction in number of elements coupled with constancy of link values (synaptic weights).

1 Introduction

Aritificial Neural Networks have been studied for many years with a two-fold goal in mind : achieving human-like performance in areas where standard computers fail or perform poorly (i.e. - image and speech recognition) and understanding the human mind better. An artificial neural network is defined as an array of simple computational elements (often termed "neurons") interconnected by links (often termed "synapses") operating in parallel. Artificial neural net models are specified by the net topology, node (neuron) characteristics and training or learning rules (usualy associated with the synapses) [LIP87]. Nets are considered to have a great many number of neurons. For example, the human brain, which is the largest known biological neural network, is presumed to have $10^{10} - 10^{12}$ neurons.

The massive parallelism achieved by artificial neural networks is only one of the benefits they present. Due to the large number of nodes ,each with primarily local connections ,these nets are usually much more robust or fault tolerant than sequential von Neumann computers. Another major benefit is the fact that a neural network adapts or learns. This is extremely important in areas involving unknown a-priori environments or dynamic environments (e.g. - new words encountered in speech recognition). Neural networks also make weaker assumptions on the environment.

Although it is commonly presumed that biological neural nets are composed of "simple" elements we cannot be sure at all of this. The biological neuron has been explored with intense vigor and there exists today quite an impressive body of knowledge concerning this elemental unit. We cannot, however, using the present knowledge, be certain that the biological neuron is indeed simple.

In contrast to biological neural nets, artificial neural nets have been defined using extremely simple elements which really are simple in a mathematical sense. Most artificial neurons sum N weighted inputs and pass the result through a non-linearity . This, however, is not necessarily the appropriate approach. We can assume that biological neurons actually carry out higher order functions of their inputs , and design networks consisting of these elements (see [GGM88] and [KOC87] for examples). Even if the end product (i.e. the network) does not resemble a biological network, it is still possible to have some computational benefits by using the basic organization of a neural network, with more powerfull elements. In this work we present artificial elements that are most generaly defined, and denote them **Teurons**.

The *analog* teuron is defined as follows:

- $X(t) = (x_1(t), \ldots, x_n(t))$ is the input vector at time t.
- $W(t) = (w_1(t), \ldots, w_k(t))$ is a dynamic coefficient vector, whose value is given at time t.
- $O(t)$ is the output of the teuron at time t defined as - $O(t+1) = f_1[f_2[\ldots\ldots[f_l(X(t), W(t))]\ldots.]]$ where $f_1, f_2, \ldots, f_l$ are arbitrary continuous or non-continuous functions.

This is a natural expansion of the formal neuron, which is easily accomodated by our model. The formal neuron is defined as:

- $X(t) = (x_1(t), \ldots, x_n(t))$ is the input vector at time t.
- $W(t) = (w_1(t), \ldots, w_k(t))$ is a dynamic coefficient vector, generally referred to as synaptic weights.
- $W(t)$ is usually represented as : $(w_{11}, w_{12}, \ldots, w_{1n}, \ldots,$
- $f_2 = \sum_{ij} w_{ij} x_j$
- f_1 is some non-linear function, e.g. a hard-limiter.

  The *digital* teuron shall be defined most generally as a Turing machine (actually a Teuring machine...).

CH2898-5/90/0000/0433$01.00 © 1990 IEEE

In the following we shall introduce two teuron networks which may be used as content addressable memories and pattern classifiers. The first network is based on the Gödel encoding and the second network is based on binary encoding.

2 The Gödel encoding

We shall use Gödel's method of coding a sequence of numbers into a single number so that the number can be decoded back into the original sequence [GOD31]. This method is described in [FIF87].

Suppose we have a sequence of non-zero integers: $a_1, \ldots, a_n$. We can form the product: $N = 2^{a_1}3^{a_2}5^{a_3}\ldots$, where $2, 3, 5, \ldots, p_n$ are the first n prime numbers. N is the code number of the original sequence and since it was formed using primes raised to a power, we can uniquely determine the power of each prime.

The sequence, coded as the number N, may be recovered using [MOS52]:

$$a_i = W(p_i, N) - 1$$

where $W(x, y)$ is the least integer z such that y is not divisible by x^z.

3 The Power Teuron

The power teuron we shall use is of the analog kind. It has n inputs : $x_1, \ldots, x_n$, n synaptic weights : $w_{1j}, \ldots, w_{nj}$,a threshold Θ_j and an output y_j.
y_j is given by:

$$O_j = \prod_{i=1}^{n} w_{ij}^{x_i}$$
$$y_j = F(O_j - \Theta_j)$$

where F is some non-linearity.

Thus, the power teuron multiplies powers instead of summing products (like the formal neuron [MCC43]).

We shall now proceed to demonstrate the application of the power teuron in building content addressable memories (CAM).

4 A simple CAM using Power Teurons

The Gödel encoding may be used to compress an entire sequence into a single number. Thus, memory of a sequence is reduced to memory of the single number. We can use this idea to build a simple CAM using the above encoding. The implementation will be based upon the analog teuron introduced in the previous section : the power teuron.

The basic topology of the network is given in figure 1.

There are n binary inputs to the network. Their values may be either "1" or "2" (the Gödel encoding uses positive integers). The number of outputs is m. Each teuron represents one sequence. Thus , in order to memorize m sequences we need m teurons. The network operates according to the following algorithm :

1. Initialize weights and thresholds:

$$w_{ij} = p_i, 1 \leq i \leq n, 1 \leq j \leq m$$

where p_i is the ith prime factor, and w_{ij} is the weight from input node i to output node j.

$$\Theta_j = \prod_{i=1}^{n} w_{ij}^{x_{ij}}, 1 \leq j \leq m$$

where $(x_{1j}, \ldots, x_{nj})$ is the jth sequence we wish the network to memorize.

2. Sequence recognition :
   (a) An arbitrary sequence $x_1, \ldots, x_n$ is given to the network.
   (b) Each teuron computes :

$$O_j = \prod_{i=1}^{n} w_{ij}^{x_i}$$

$$y_j = F(O_j - \Theta_j)$$

where :

$$F(x) = \begin{cases} 1 & \text{x=0} \\ 0 & \text{otherwise} \end{cases}$$

As the coding is unique only one teuron will output 1 - if the network has memorized this sequence. Otherwise, all the teurons will output 0.

This network ,although simple ,has some advantages when compared to other types of neural network CAMs. It is feed forward ,as opposed to many CAMs which are iterative. Such networks may iterate many times until convergence, which is not always guaranteed. Our network does not iterate at all (thus supplying an answer much faster) and "convergence" is always guaranteed. It also requires exactly one teuron per memorized sequence. Thus , it is minimal in the sense that any network with less than m elements will not be completely parallel. The fact that we have exactly one teuron per sequence may seem to affect the network's fault tolerance (teuron destruction means memory loss). However, this is not a difficult issue to resolve. We shall discuss this point further ahead.

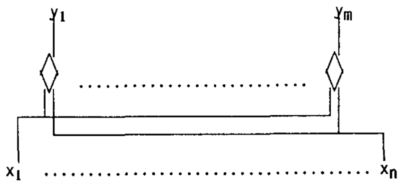

Figure 1: topology of a simple power teuron memory. Teurons depicted by ◇

5 A pattern classifier using Gödel encoding

We shall now present an enhancement to the above scheme, namely a network that will be able to classify n bit inputs into m distinct classes. The network will use Gödel encoding to "remember" the exemplar for each class. Each input will be compared to each of the m exemplars using Euclidian distance (which is actually the hamming distance in binary inputs). The class chosen will be that with the minimal hamming distance between the exemplar and the input pattern. The topolgy of the network is given in figure 2.

The network works according to the following algorithm :

1. Initialize weights and thresholds: $w_{ij} = p_i$ for $1 \leq i \leq n$, $1 \leq j \leq m$
   where p_i is the ith prime factor and w_{ij} is the weight from input node i to output node j.
   $$\Theta_j = \prod_{i=1}^{n} w_{ij}^{x_{ij}}, 1 \leq j \leq m$$
   where $(x_{1j}, \ldots, x_{nj})$ is the exemplar of class j.
2. Present an input pattern to the network : $x_1, \ldots, x_n$
3. Compute outputs (i.e. compute y'_j) :
   $$y'_j = \sum_{i=1}^{n} [g(x_{ij}) - x_i]^2$$
   where $x_{ij} = R(\Theta_j / w_{ij} * w_{ij})$
   $R(x/y)$ is the remainder of x divided by y.
   $$g(x) = \begin{cases} 2 & \text{x=0} \\ 1 & \text{otherwise} \end{cases}$$
4. Pick min teurons (described in the following section) compute the minimum y_j. Self min teurons (also described) may also be used.

The network initializes the weights to prime numbers and the thresholds to the Gödel number representing the appropriate exemplar. The network then compares the the input pattern to each exemplar by decomposing the exemplar back into the original bits. The results of the comparisons (m comparisons for m exemplars) are then passed through one or more self-min teurons.

A self-min teuron receives all the values of the other teurons (in our case all the other hamming distances) and outputs 1 only if it is the minimum otherwise it will output 0.

We may implment this teuron as a digital teuron. As such ,it will use extremely simple circuits known as - comparators. Such circuits are used in ALUs and in associative memories. Comparators may work in parallel, thus achieving high performance.

This teuron may also be implemented as an analog teuron in the following manner: suppose it receives $M - 1$ values - $x_1, \ldots, x_{M-1}$ and its own value is x_s (standing for x_{self}). The teuron computes the following function:

$$\begin{aligned} f = \quad & [(x_1 - x_s) + (x_2 - x_s) + \ldots + (x_{M-1} - x_s)] \\ - \quad & [|x_1 - x_s| + |x_2 - x_s| + \ldots.. + |x_{M-1} - x_s|] \end{aligned}$$

We then pass the result through the following nonlinear function g :

$$g(x) = \begin{cases} 1 & \text{x=0} \\ 0 & x \leq 0 \end{cases}$$

(note : $f \leq 0$).

A pick-min teuron receives M values , selects the minimum one and triggers the appropriate output.

A digital teuron implementing the pick-min function will use comparators working in parallel like the ones we used in the self-min digital teuron. An analog implementation will simply compute the function $g \circ f$ presented in the self-min scheme for each input x_i , and then select the x_i which produces an answer of "1".

We shall now present one more scheme used for CAM and pattern classification. A discussion of both schemes will then follow.

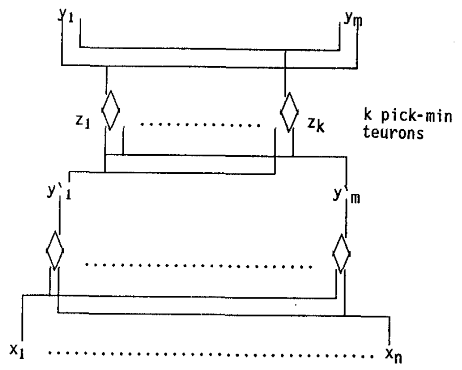

Figure 2: topology of a pattern classifier

6 Using binary encoding in CAM and pattern classification

While the Gödel encoding is general and may be used for any sequence of numbers there is a simpler encoding we may use for binary numbers : binary encoding. Thus, we treat each n bit input pattern as an n bit binary number. Obviously , each number represents exactly one pattern.

This encoding is much simpler to obtain and it may be used instead of the Gödel encoding in the above schemes for CAM and pattern classification. We shall now present the algorithm for pattern classification using binary encoding. The algorithm for the CAM network may be obtained in a straightforward manner. The topologies are exactly those presented above. The bit values are "0" and "1".

The algorithm for pattern classification using binary encoding :

1. Initialize weights and thresholds :

$$w_{ij} = 2^{i-1}, 1 \leq i \leq n, 1 \leq j \leq m$$

$$\Theta_j = \sum^{n} w_{ij} x_{ij}$$

   where $(x_{1j}, \ldots, x_{nj})$ is the exemplar of class j.

2. Present an input pattern to the network : $x_1, \ldots, x_n$
3. Compute outputs (i.e. compute y'_j) :

$$y'_j = \sum_{i=1}^{n} (x_{ij} - x_i)^2$$

   where
   $x_{nj} = g[\Theta_j - w_{nj}]$
   $x_{(n-1)j} = g[\Theta_j - w_{nj} x_{nj} - w_{(n-1)j}]$
   $x_{(n-2)j} = g[\Theta_j - w_{nj} x_{nj} - w_{(n-1)j} x_{(n-1)j} - w_{(n-2)j}]$
   $\vdots$

$$g(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

4. Pick min teuron (or teurons) compute the minimum y_j. Self min teurons may also be used.

As we can see the teuron computing the binary encoding is quite simple using only sums and products. The result of phase 3 which is the hamming distance of each exemplar from the input pattern is passed on to one or more pick-min (or self min) teurons. Thus, the class with minimum hamming distance is chosen. We shall now discuss the networks presented in the paper.

7 Discussion

The networks presented above have a few advantages over similar types of networks used for memory and pattern classification.

The networks are essentially feed forward and thus convergence is always guaranteed. Networks such as the Hamming net [LIP87] [LGM87] and the Hopfield net [HOP82] are iterative and convergence is not always guaranteed. No iterations also means, in this case, faster convergence. It should be noted that there are other neural network models which are feed forward and have these advantages. Such models include the Multi-layer perceptron [LIP87] [MIP69] used for pattern classification and Kohonen's self organizing feature maps used as associative memories [LIP87] [KOH84].

A very important point when implementation issues are concerned, lies in the fact that the weights are essentially constant (primes in the Gödel network and powers of 2 in the binary encoding network). Thus when implmenting such a network we can use simple electronic elements which may be hardwired and non-changable. For example if we use resistors for the synapses then they may be constant resistors and not varying resistors. This fact may decrease the cost of such a network. Instead of n varying weights per neuron in neural networks we have one varying threshold.

Comparing the two networks amongst themselves we note that the Gödel network may reach extremely large thresholds very fast. This is due to the fact that we multiply primes and thus the numbers "behave" in a manner similar to the function $n!$. The binary encoding network solves this problem by using an encoding which, although suitable for binary numbers only, is much more conservative in the values of thresholds.

The Gödel network's main advantage lies in the fact that the scheme is general. We may use continuous input values (altough this would require a stronger teuron because the one we presented relied on the fact that there are only two input values possible, when decomposing the Gödel number).

If we view the input as a binary number then the Gödel encoding may produce a code number which is larger , in some cases, than the original series (e.g. the input "$1, 1, 1$" which has a value of 7 produces a Gödel number of $2 * 3 * 5 = 30$. This is not always the case. Consider , for example, a picture whose elements are 8 bit pixels - that is pixels with values in the range $[0, 255]$. Thus ,the elements are actually digits in base 256. Suppose, further ,that the intensity histogram of the image is bimodal, with a small range of intensity values as the background. Then , a 2 pixel sub image might have a value of ,say ,"$2, 1$" which translates to $2 * 256 + 1 = 513$ with a Gödel code of $2^2 * 3 = 12$). We must remember, however, that we have moved to an analog representation. Thus , theoretically at least, we have no limitations on the size of the number to be stored. The Gödel encoding is also unique (i.e. each sequence is represented by one number which represents that sequence only) whereas other codes are not neccassarily so. Another point to note is the fact that the "size" of a number is something which can easily be tailored according to need. In our case we could , as the Gödel number is an integer, map this number into the range $[0, 1]$.

The Gödel code also has the advantage of being "sparse", in the sense that code numbers are set apart from each other. This may help in situations of "drifting" of the memory. Thus, for example, if we take the sequences "1,1,1" and "1,1,2" (where "1" ,"2" actually represent "0" and "1" respectivley) then their binary values are 0 and 1 while their Gödel encoding is 30 (2*3*5) and 150 ($2 * 3 * 5^2$). Thus , if the element memorizing these numbers "drifts" by one unit then in the binary encoding case we have a completely different number. In the Gödel case we are still in the "safe" range. In view of the arguments presented in this paragraph we maintain that our method does indeed have merit.

Current technology has been able to deal with extensive analog elements. Dynamic RAMs of four Mbits exist which actually consist of four million analog elements. Although each element uses essentially two voltage levels ("0" and "1" levels), it is capable of many more. CCD cameras exist which employ an array of 1000 x 1000 analog elements, each one capable of translating light energy into one of a 1000 levels of electrical energy. Thus we have one million analog elements capable of 1000 energy levels. This is typical of todays analog elements - their voltage resolution is in the range of milli-volts.

Using , for example, elements capable of 1024 levels we achieve a reduction of 10 to 1 in the number of elements required. A 10 bit number needs one such element instead of 10 bit elements. A one million array of 1024 level elements may be used to store 10000 100-digit numbers of base 1024. Thus, we may store 10000 numbers of up to 1024^{101} or 2^{1010}.

It should be noted that analog elements have problems (which also hinder many neural network implementations). They are less stable than digital elements and need constant refresh (e.g. in the CCD camera the image must be constantly refreshed). Also, the industry seems to have moved in the digital direction , which is easier to handle. We argue that analog implementations present intriuging possibilities. Future research may come up with analog elements which are easy to handle, and which may be miniaturized to such an extent that valuable chips may be built.

The fact that our network is confined to a limited number of memorized sequences seems at first sight as a drawback, comparing to other types of neural networks. However, this is not the case: the same holds for any type of Hopfield like network - only that the limiting factor is around $0.15n$, where n is the size of the network in the original model, or a somewhat higher limit in related models. Another point related to this issue is the degree of fault tolerance required from such a network. This issue has often been cited as one of the main advantages of neural networks. We suggest that much less emphasis should be put on the issue of fault tolerance in such network. . While biological neural nets use unreliable elements (neurons and synapses) which may be destroyed (thousands of neurons die each day in an adult human), artifical nets use extremely reliable electronic or optic elements. Thus , while trying to gain from understanding the principles of biological neural computations, it might not be necessary to simulate its redundancy, as well as other non (computationally) functional details.

References

[1] [FIF87] Fischler M.A. & Firschein O. , "Intelligence : The Eye, The Brain and the Computer", Addison-Wesley, 1987.

[2] [GGM88] Giles C.L., Griffin R.D and Maxwell T., "Computational Advantages of Higher Order Neural Networks", Proc. 1st INNS meeting, Boston 1988.

[3] [GOD31] Uber Formal Unentscheidbare Satze der Principia Mathematica und Verwandter Systeme, 1. Monatshefte Math. Phys., Vol. 38, pp 173 - 198, 1931.

[4] [HOP82] Hopfield J.J. , "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc. Natl. Acad. Sci. Usa, Vol. 79 , pp 2554-2558, April 1982.

[5] [KOH84] Kohonen T., "Self Organization and Associative Memory", Springer - Verlag, Berlin, 1984.

[6] [KOC87] Koch C., "A Network model for Cortical Orientation Selectivity in Cat Striate Cortex", Invest. Ophtalmol. Vis. Sci. 28 (Supl 3), 126, 1987

[7] [LGM87] Lippmann R.P. & Gold B. & Malpass M.L., "A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification", MIT Lincoln Laboratory Technical Report TR-769, 1987.

[8] [LIP87] Lippmann R.P., "An Introduction to Computing with Neural Nets", IEEE ASSP magazine, April 1987.

[9] [MCC43] McCulloch W.S. & Pitts W., "A Logical Calculus of the Ideas Imminent in Nervous Activity", Bull. Math. Biophysics, 5, 115-133, 1943.

[10] [MIP69] Minsky M. & Papert S., "Perceptrons: An Introduction to Computational Geometry", MIT Press 1969.

[11] [MOS52] Mostowski A., "Sentences Undecidable In Formalized Arithmetic - An Exposition of the Theory of Kurt Gödel", North-Holland Publishing Company, Amsterdam, 1952.