ICNN '94

Proceedings of IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS

June 28 - July 2, 1994

IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE

IEEE

ICNN WCCI

Steepest Descent Retrieval Algorithm for Autoassociative Neural Memory

Bogdan M. Wilamowski
University of Wyoming
Dept. of Electrical Engineering
Laramie, WY 82071

Jacek M. Zurada
University of Louisville
Dept. of Electrical Engineering
Louisville, KY 40292

Aleksander Malinowski
University of Louisville
Dept. of Electrical Engineering
Louisville, KY 40292

Abstract

The proposed steepest descent retrieval algorithm is shown to improve the recovery of stored patterns in autoassociative recurrent neural memories. The algorithm implements the steepest reduction of the computational energy function rather than an ordinary random and unqualified update. The experiments indicate that this update mode is the more efficient when compared to the conventional asynchronous update. Specifically, it performs better in recovery of vectors at smaller Hamming distance and in rejection of stable but spurious memories.

Introduction

Consider a fully connected autoassociative neural memory consisting of *n* bipolar neurons. Assuming that the memory network updates in a discrete-time mode and its outputs are one of the 2^n bipolar binary *n*-tuple vectors, considerable insight into its performance can be gained by evaluating its energy function. Notably, the asynchronous recurrent update never increases the energy, and the state transitions terminate in one of the local energy minima located at cube vertices.

The memory updates at the i-th output are produced by the asynchronous vote of all neurons but the i-th. Intuitively, the degree of "incorrectness" of the i-th neuron should be given priority so that the bits with higher probability of error are updated first. Obviously, the sequence of updates produced by this method is rather essential since once the i-th bit has been corrected, it later participates in subsequent "votes" on bits other than itself.

One natural measure of the probability of error of the i-th memory bit is the value of the activation level at its neuron input. The larger its absolute value is for the neuron that will be updating, the larger will be the resulting decrease of the network energy performed by this update. This paper makes use of the steepest decent energy update scheme, thus introducing the error correcting scheme for bits with high probability of error. This, in turn, improves the efficiency and robustness of retrieval process.

Storage Algorithm and Energy Function

Autoassociative memory involves single layer neural network with feedback as it is shown in Figure 1 [1]. Its architecture is based on the Hopfield network [2]. For this memory patterns are encoded using the simple method of computing the autocorrelation matrix for each of the bipolar binary pattern s_i to store. The weight matrix is then computed using the superposition principle

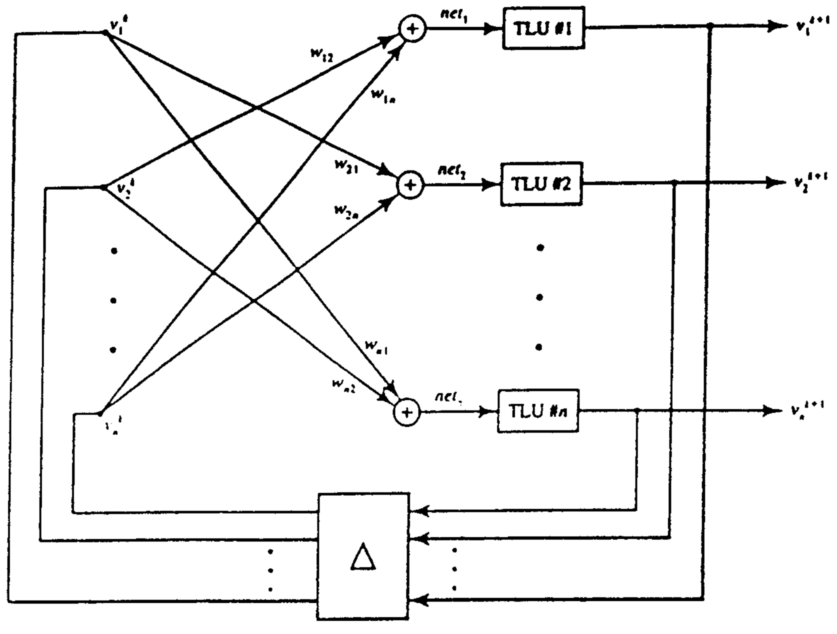
$$W = \sum_{m=1}^{p} s_i s_i^t - pI \tag{1}$$

or, in matrix notation

$$W = SS^t - pI \tag{2}$$

where S is the rectangular $p{*}n$ matrix encoding p patterns of size n. Matrix W is symmetrical with zero elements on the main diagonal. The energy function for such system can be expressed as [3]:

$$E(v) = -\frac{1}{2} v^t W v \tag{3}$$



(a)

Fig. 1. Network structure of autoassociative memory

Note that the simple storage algorithm (1) or (2) results in a network without self-feedback for individual neurons and without the biasing weights adjusting the threshold. Such network structure is closely related to the definition of the network's energy given by equation (3). The system energy for a given vector *v* can also be computed as a sum of energies for each pattern to be stored

$$E(v) = \sum_{i=1}^{p} E_i(v) \tag{4}$$

where

$$E_i(v) = -\frac{1}{2} v^t \left(s_i s_i^t - I\right) v \tag{5}$$

Note that for binary bipolar vector we have $v^t I v = n$, where n is the size of the network layer. Also $v^t s = n - 2\,HD(v,s)$, where $HD(v,s)$ is the Hamming distance between binary vectors v and s. Each component of the energy function given by equation (5) can therefore be expressed as a function of the Hamming distance HD between a stored
pattern s_i and a given vector v as

$$E_i(v) = -\frac{1}{2}\left[\left(n - HD(v,s_i)\right)^2 - n\right] \tag{6}$$

where n is the size of vectors v and s_i. As a result, each component of the energy function (6) has a shape of an inverted parabola with minima at s_i and at the complement of s_i, $i=1,2,..p$ as it is shown in Figure 2. Combining equations (4) and (6) leads to

$$E(v) = -\sum_{i=1}^{p}\left[n^2 - n + 4\,HD^2(v,s_i) - HD(v,s_i)\right] \tag{7}$$

or

$$E(v) = 2\sum_{i=1}^{p}\left(n - HD(v,s_i)\right)HD(v,s_i) - \frac{pn(n-1)}{2} \tag{8}$$

Obviously, when many patterns are stored, a rather complicated energy surface is formed with many local minima. Those local minima are produced at the locations of stored patterns s_i and at its complements. At the same time, many spurious local minima of the energy function are formed. Their presence is seriously detrimental for memory's performance.

Steepest Descent Retrieval Algorithm

The encoded information is retrieved form memory by applying an initializing vector to the input and letting the network compute the output. Then, the updated output is supplied to the input (see Figure 1) and a new output is computed. This process is repeated until the steady-state condition is reached. This "synchronized" updating very often leads to oscillations between two or more states with the same energy, and a local energy minimum may never be reached [4]. In order to avoid these local oscillations, the "asynchronous" updating has been introduced. At one cycle, only one randomly chosen output is allowed to change under this update rule. This sequence of updates converges to the one of the minima without oscillations.

The "asynchronous" updating has the following disadvantages:

1. It is slow since randomly chosen output is not always the one leading to the energy minimum via the shortest path.
2. When two neighboring minima with the Hamming distance of unity have the same energy value, the asynchronous updating process typically enters oscillations.
3. Some stored memory states with "weak" minima of energy surface may be nonrecoverable.

In order to eliminate some of the disadvantages of a conventional asynchronous updating the following modification is proposed.

Instead of randomly choosing an output for updating, the output with maximum gradient of the energy function is chosen. This way the retrieval process requires less steps and also the retrieved pattern is closest (most similar) to the applied vector v. The standard asynchronous updating process, in many cases, may lead to a pattern which is very far from the applied input. Indeed, since the energy gap between the initializing pattern and its closest stored prototype vector is fixed, updating with large energy increments naturally leads to the retrieval of vector which is closest to the key vector in the Hamming distance sense. The gradient of the energy function is given by

$$\nabla_v E(v) = - Wv \tag{9}$$

and each of the components of the negative gradient vector is computed at each neuron

$$net_i = w_i v \tag{10}$$

where w_i is the vector of weights fanning in the i-th neuron. Using equation (1) and knowing that $s'v = n - 2\,HD(v,s)$, equation (9) can be rewritten in the form:

$$\nabla_v E(v) = pv - \sum_{i=1}^{p} s(n - 2\,HD(s,v)) \tag{11}$$

Obviously, the output state can only change when its product of $net_i\, v_i$ is negative. Under the steepest gradient descent rule the strongest "driving power" for updating has the neuron with the smallest value of $net_i\, v_i$, and this neuron is chosen for updating. Only in the case when two or more neurons have the same value of $net_i v_i$ the output chosen randomly between two. The updating process continues while

$$\max_{i=1,\dots,n}\left(net_i v_i\right) \le \alpha_{th} \tag{12}$$

is valid. In simplest case α_{th} is assumed to be zero. The updating process is terminated if the lowest of all $net_i\ v_i \geq \alpha_{th}$ reaches the positive value. There is a further possibility for modification of the updating process by setting some negative value to α_{th} for earlier termination of the updating process. This way improved pattern recovery can be obtained. For example, if many patterns are stored in the memory and two of them are close together (their Hamming distance is equal one) and the energy for these two patterns is different, then usually only one pattern with the smaller energy is recovered. However, by setting negative value to α_{th}, the updating process can be terminated if the output vector reaches the stored pattern with the larger energy function. Since in the proposed updating scheme in each cycle the output changes within the Hamming distance of one and it uses the "shortest way" for choosing the neuron. The recovered pattern is therefore always the closest to the input pattern in the Hamming distance sense.

Example 1

```
stored patterns
#1    1 -1 -1  1    E= -4.0    HD= 0  2
#2   -1 -1  1  1    E= -4.0    HD= 2  0

weights
     0     0    -2     0
     0     0     0    -2
    -2     0     0     0
     0    -2     0     0

number of retrieved patterns:
             #1  #2 #1c #2c spurious
asynchro. 245 254 250 251      0
grad(0)    265 249 261 225      0
grad(-1)   290 236 251 223      0
grad(-2)    65  52  79  55    749
```

Example 2

```
stored patterns
#1   -1 -1  1  1    E= -6.0    HD= 0  1
#2   -1  1  1  1    E= -6.0    HD= 1  0

weights
     0     0    -2    -2
     0     0     0     0
    -2     0     0     2
    -2     0     2     0

number of retrieved patterns:

             #1  #2 #1c #2c spurious
asynchro.    0 612 387   0      1
grad(0)    266 220 264 250      0
grad(-1)   285 239 243 233      0
grad(-2)   250 259 244 247      0
```

Example 3

```
stored patterns
#1    1 -1 -1  1    E= -4.0    HD= 0  2  3
#2   -1 -1  1  1    E= -4.0    HD= 2  0  1
#3   -1  1  1  1    E= -6.0    HD= 3  1  0

weights
     0    -1    -3    -1
    -1     0     1    -1
    -3     1     0     1
    -1    -1     1     0

number of retrieved patterns
             #1  #2  #3 #1c #2c #3c spurious
asynchro.    2   1 498   1   0 498      0
grad(0)      0   0 495   0   0 505      0
grad(-1)   203 130 167 199 149 152      0
grad(-2)   188 129 164 202 150 167      0
```

Example 4

```
stored patterns
#1    1 -1 -1  1    E= -4.0    HD= 0  2  3  3
#2   -1 -1  1  1    E= -4.0    HD= 2  0  1  3
#3   -1  1  1  1    E= -4.0    HD= 3  1  0  2
#4   -1  1 -1 -1    E= -4.0    HD= 3  3  2  0

weights
     0    -2    -2     0
    -2     0     0    -2
    -2     0     0     2
     0    -2     2     0

number of retrieved patterns
            #1  #2  #3  #4 #1c #2c #3c #4c spurious
asynchro.   1   0 391   0   1 230   0 377     0
grad(0)   192  52 159  67 216  55 188  71     0
grad(-1)  188  71 164  73 202  77 167  58     0
grad(-2)  184  74 160  70 211  66 185  50     0
```

Example 5

```
stored patterns
#1  1  1  1  1 -1 -1  1  1  1   E= -24.0   HD= 0  3
#2  1 -1  1  1  1 -1  1 -1  1   E= -28.0   HD= 3  0

number of retrieved patterns
            #1  #2 #1c #2c spurious
asynchro. 234 168 156 202   240
grad(0)   249 271 239 241     0
grad(-2)  167 170 176 163   324
```

Example 6

```
stored patterns
#1  1  1  1  1 -1 -1  1  1  1   E= -24.0   HD= 0  3  5  4  4
#2  1 -1  1  1  1 -1  1 -1  1   E= -28.0   HD= 3  0  6  5  5
#3 -1  1 -1 -1  1 -1 -1  1  1   E= -40.0   HD= 5  6  0  3  7
#4  1  1  1 -1  1 -1 -1  1 -1   E= -24.0   HD= 4  5  3  0  4
#5  1 -1  1  1 -1  1 -1  1 -1   E= -32.0   HD= 4  5  7  4  0

number of retrieved patterns
             #1  #2  #3  #4  #5 #1c #2c #3c #4c #5c spurious
asynchro.    78  16 132  72  44   9   8 181  56  86   318
grad(0)      39   0 334  56  60  43   0 335  49  84     0
grad(-2)     30  39 146  23  51  29  65 137  28  67   385
grad(-4)     12  41  79  19  49  11  25  96  25  50   593
grad(-6)      6  11  54   4  32   9  13  53   5  36   777
grad(-8)      7   8  28   2  10   6   6  30   3  13   887
```

Examples - All retrieval processes were performed with 1000 randomly chosen initial patterns. The following symbols are used: E - energy, HD - Hamming distance between stored patterns, #1, #2, ... #p - stored patterns, #1c, #2c, ... #pc complements to the stored patterns, asynchro. - conventional asynchronous updating, grad(α_{th}) - gradient based method different α_{th} parameters.

Experimental Results

In order to illustrate the performance of the retrieval algorithm six computational examples are shown below. In all cases patterns were stored in the memory using autocorrelation encoding as described earlier. For the first two examples, a four dimensional network is used with two stored patterns. The Hamming distance between stored patterns equals $HD(s_1,s_2) = 2$ in the first example, and in the second example $HD(s_1,s_2) = 1$. In all examples 1000 randomly chosen initial patterns were used and both conventional asynchronous and gradient based retrieval methods were used. Different values od α_{th} were used for gradient updating. The updating process converges to stored patterns (#1, #2, ... #p), to complements of stored patterns (#1c, #2c, ... #p) or to spurious states.

In case when only two patterns with $HD = 2$ are stored both "asynchronous" updating and "gradient based" updating methods are working properly (Example 1). In case when $HD = 1$ the asynchronous updating fails, but the proposed retrieval algorithm works correctly (Example 2). When more patterns are stored, the memory becomes heavily overloaded and "asynchronous" updating is able to recover only a limited number of patterns (Examples 3 and 4). Note that by adjusting the threshold parameter α_{th} one can control the number of patterns which can be recovered. In Example 3, pattern #2 has the energy value $E=-4$ and pattern #3 has the energy $E=-6$. Since the Hamming distance between patterns #2 and #3 is of unity value, pattern #2 is not at local minimum. The proposed algorithm with $\alpha_{th} = -2$ is able to recover this pattern. Similarly in the same example pattern #3 is within the Hamming distance of one to the complement of #1, and asynchronous updating has difficulty recovering it.

The other two examples are related to the nine dimensional network with two and five patterns stored. With larger network sizes, the conventional asynchronous algorithm is not even able to properly recover each of two stored patterns. In Example 5, for instance, asynchronous updating converge to spurious states in 24% of the cases cases. When the gradient based method is used with $\alpha_{th} = 0$, all stored patterns are correctly recovered. A similar result was obtained in Example 6 with five patterns stored.

Conclusions

The proposed retrieval algorithm requires only a reduced number of updating cycles approximately equal to the Hamming distance between input and the retrieved pattern. For small networks it is three to ten times faster than "asynchronous" updating. For a larger network the convergence is even better. The problems of convergence to spurious states under the conventional update scheme have been emphasized in the paper by us using overloaded memory. This has been done deliberately both to magnify the difficulties encountered with the conventional update scheme and to illustrate the benefits of the proposed approach.

The algorithm allows for gradual control of the termination of the updating process by changing the α_{th} parameter. Therefore allows it to recover patterns stored in the memory which are not necessily at the local energy minimum. Very promising results were also obtained when the proposed algorithm was used as the retrieval algorithm in the continuous-type recurrent networks.

References

[1] J. M. Zurada, "Introduction to Artificial Neural Systems", West Publishing Company 1992.

[2] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proc. National Academy of Science, 79, pp. 2254-2558, 1982

[3] D. W. Tank, J. J. Hopfield, "Simple neural optimization networks: An A/D converter, signal decision circuit and a linear programming circuit," IEEE Trans. Circuits Syst., vol. CAS-33, May 1986, pp. 533-541.

[4] M. Hasler, "Recursive neural networks for associative memories," John Wiley, London 1990.