Progress Report
February 16 2011 - March 7 2011

Raj K. Kumar, Amir Hesam Salavati
E-mail: raj.kumar@epfl.ch, hesam.salavati@epfl.ch

Supervisor: Prof. Amin Shokrollahi
E-mail: amin.shokrollahi@epfl.ch

Algorithmics Laboratory (ALGO)
Ecole Polytechnique Federale de Lausanne (EPFL)

March 8, 2011

1 Introduction

In this report, we will discuss a new approach to increase the pattern retrieval capacity of neural associative memory. The suggested method is based on *non-binary* neurons in a hetero-associative neural network with *asymmetric* weights. We will show that if we select weights properly, we will be able to store an exponential number of patterns in terms of the size of network.

In what follows, we will first explain the model more clearly. A proper way to select weights, which is based on lattice codes in coding theory, is discussed next. Some simulation results are provided as well to evaluate the proposed approach. Finally, we suggest some extensions to the current model which might improve the performance.

2 Model

We consider a hetero-associative neural network which can be modeled by a bipartite graph with a set of K nodes on one side and N nodes on the other side. We call the first set, composed of the nodes $z_1, \ldots, z_K$, *message nodes* and the other set, with N nodes $y_1, \ldots, y_N$, the *code nodes*. Figure 1 illustrates an example of the bipartite graph. In the neural graph, each node

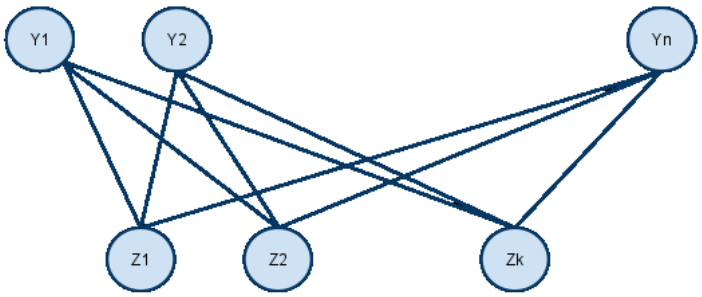

Figure 1: A hetero-associative neural network

represents a neuron with multi-level integer states. In this model, instead of taking the firing condition of the neurons as their states, we count the number of spikes fired by each neuron in a given time window and take that

as the neural state. Therefore, the state of a neuron in this model could be a non-negative integer number. Furthermore, since the short-term firing rate is limited in real systems, we also consider the state of neurons to be bounded to some reasonable value.

Similar to other neural models, neurons update their state based on the feedback from other neurons. Here, we assume that the state of neuron i, which we denote by s_i, is a function of the weighted sum of other neurons' states:

$$s_i = f(\sum_{j \in N} w_{ij} s_j),$$

In which $f(.)$ can be any function with non-negative bounded integral support. In this report, we take $f(.)$ to be a *saturated linear* function. More specifically, let h_i^z and h_l^y denoted the weighted sum of the i^{th} message and the l^{th} code neurons, i.e.

$$h_i^z = \sum_{j=1}^{N} W_{ji} s_j^y$$

and

$$h_l^z = \sum_{j=1}^{K} G_{jl} s_j^z$$

where s_j^z and s_j^y are the states of the j^{th} message and code neurons, respectively, W_{ji} is the weight from y_j to z_i and G_{jl} represents the weight from z_j to y_l. The weights are asymmetric and can take any real (reasonably bounded) number.

Having defined the weighted sum over input links, we assume the state of the i^{th} message neuron is updated according to equation (1). Similarly, equation (2) shows the update rule fo the state of the k^{th} code node.

$$s_i^z = \begin{cases} z_{max}, & \text{if } h_i^z \geqslant z_{max} \\ z_{min}, & \text{if } h_i^z \leqslant z_{min} \\ round(h_i^z), & \text{Otherwise} \end{cases} \quad (1)$$

$$s_l^y = \begin{cases} y_{max}, & \text{if } h_l^y \geqslant y_{max} \\ y_{min}, & \text{if } h_l^y \leqslant y_{min} \\ round(h_l^y), & \text{Otherwise} \end{cases} \quad (2)$$

In the above equations, $round(.)$ returns the closest integer value, z_{max} and z_{min} are the maximum and minimum values for the state of each message neuron. y_{max} and y_{min} represent similar values for code neurons.

In summary, the model specifications are:

- A bipartite graph with K message nodes on one side and N code nodes on the other ($K \leqslant N$).
- Neurons with multi-level bounded integer-valued states
- Asymmetric weights
- A saturated linear output as a function weighted input sum

3 Lattice-based Neural Associative Memory

In this section, we use the model described in the previous section and determine proper weight matrices such that the network is able to memorize an exponential number of association (in terms of K and N) between the message and code patterns. The term "lattice" in the definition comes from the similarity of the approach to lattice codes in communication systems. In short, lattice codes are a family of error correcting codes in which operations are done in real field as opposed to the binary field.

Consider a given message pattern with K elements. For simplicity, we assume that these elements can take only two values: z_{max} and z_{min}. Therefore, we have $M = 2^K$ of such patterns. We assume that the message patterns can be any of these M possible vectors uniformly at random. Letting $Z^i, i = \{1, \ldots, K\}$ be a binary vector length K which denotes the i^{th} message pattern, in the learning phase, the network **constructs** the code patterns by multiplying each of the message patterns by the $K \times N$ weight matrix G and applying the saturated linear update rule (2), i.e. $Y^i = f(Z^i G)$, where Y^i is a binary vector of length N. Since $N \geqslant K$, this operation is equivalent to going to a higher dimension and deal with noise there, which gives us better performance in presence of noise if we pick G appropriately.

During the recall phase, we assume that we have a noisy version of a *code pattern* and would like to recall the correct message pattern. For instance, let's assume we would like to recall a noisy version of pattern γ, i.e. $Y = Y^\gamma + E$, where E is the noise vector with random integer (positive and negative) elements. As a consequence of equation (1) we have to multiply Y by the code-to-message-nodes weight matrix W. Hence, the vector of input sums for message neurons, denoted by H^z will be equal to $Y^\gamma \times W + E \times W$.

For simplicity, let's assume that there is no need for saturation in update rules (1) and (2) if there is no noise. Furthermore, let's pick W such that $G \times W = I_{K \times K}$, where $I_{K \times K}$ is the $K \times K$ identity matrix. As a result of linear update rule we will have:

$$\begin{aligned} Z = f(H^Z) &= f(Y^\gamma.W + E.W) \\ &= f(Z^\gamma + E') \end{aligned}$$

Where $f(.)$ is the saturated linear update rule of (1) and E' is a linear transformation of the original noise vector E. In the above relationship, the first term, Z^γ is the desired term. Hence, we have to properly select G and W such that E' is small enough to be taken care of by the $round(.)$ operation in (1).

More specifically, the goal is to pick two weight matrices G and W such that:

1. $G \times W = I_{K \times K}$, where $I_{K \times K}$ is the identity matrix. In other words, we select W to be the pseudo-inverse of G.

2. W and G have the property that if E and E' are random vectors integer elements with length N and K, respectively, then $\| E \times W \|_2$ and $\| E' \times G \|_2$, where $\| . \|_2$ is norm-2 of a vector, be reasonably small such that the $round(.)$ operation in the neural update rules takes care of them. Another possibility is that if we continue the association process for multiple iterations back and forth, then we might ask for $\| E \times W \|_2 < \| E \|_2$ and $\| E' \times G \|_2 < \| E' \|_2$

At this point, we have not yet optimized the weight matrices according to the second requirement. However, we know that there are some good generating matrices for lattice codes in coding theory that results in lattice point with good distance properties. Therefore, a reasonable amount of noise may still be tolerated since it can not cause any confusion between lattice points. One example of such lattice generating matrix is given below for

$N = K = 8$:[1]

$$G = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3}$$

4 Simulation Results

We have simulated the proposed method for a neural network with $K = N = 8$ with the generator matrix G given by equation (3). Furthermore, the code-to-message-nodes weight matrix W is selected such that $G \times W = I_{K \times K}$.

As discussed before, we select the message patterns such that each message neuron can take either of two values: z_{max} and z_{min}. Furthermore, in order to have a realistic network, we take z_{max} to be a reasonably small value. In addition, we enforce a maximum limit, indicated by y_{max}, on the firing rate of each individual code pattern.

For the noise model, we have considered an *integer-valued* additive white Gaussian noise with zero mean and variance σ^2. We have simulated the network explained above for several values of pick Signal to Noise Ratio (SNR) defined to be $(y_{max})^2/\sigma^2$. Figure 2 illustrate the pattern retrieval error rate versus SNR for $z_{max} = 4$ and $z_{min} = 1$. Here $M = 255$. The performance is compared to a random full rank weight matrix G with the same conditions on z_{max}, z_{min} and E_{max}. As can be seen from the figures, if G is selected randomly, the error performance is very poor.

The noise model we considered above is just one of possible several options. Another reasonable choice is to evaluate the error correcting power of the method based on the number of erroneous code symbols that it can tolerate. Here, we assume that an erroneous code node has a value ± 1 to that of what it should have. The difference between this noise model and the previous case is that with the additive Gaussian noise, we might get large errors in the firing rate of a neuron. However, addition of or omitting a spike

[1] We have slightly modified the definition of matrix G to get rid of negative values in codewords.

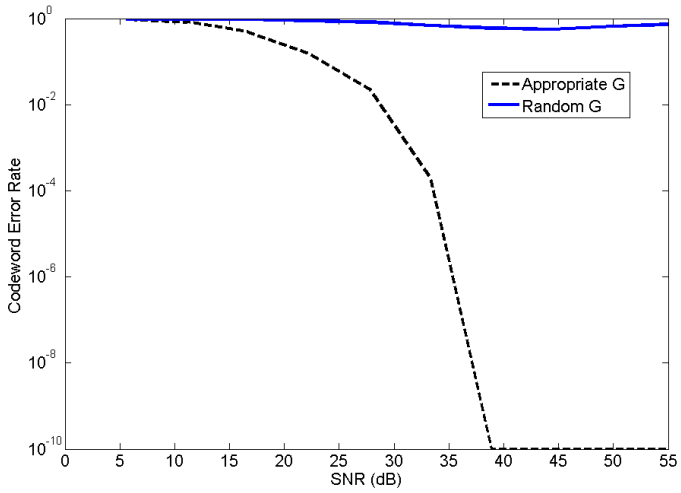

Figure 2: Pattern retrieval error rate against SNR for the proposed lattice-based approach

might be more realistic in context of neural networks.

Having simulated this scenario, we have observed that the proposed method is able to correct up to 8 code bit errors, which is equivalent to saying that the suggested method doesn't care if all the neurons skip one spike or fire one more than what they are supposed to. Simulating bigger networks is one of our future works.

Now if we assume code nodes can be erroneous with the amplitude of error being between -2 and $+2$, then figure 3 shows the pattern retrieval error rate versus the initial number of bit errors in code nodes. The performance is again compared to that of a random weight matrix. The simulation conditions are the same as before.

4.1 Extension

So far, we have assumed the state of each neuron to be a non-negative bounded integer value. However, if we allow negative values, we might be able to get better results. Furthermore, in the above simulations we have

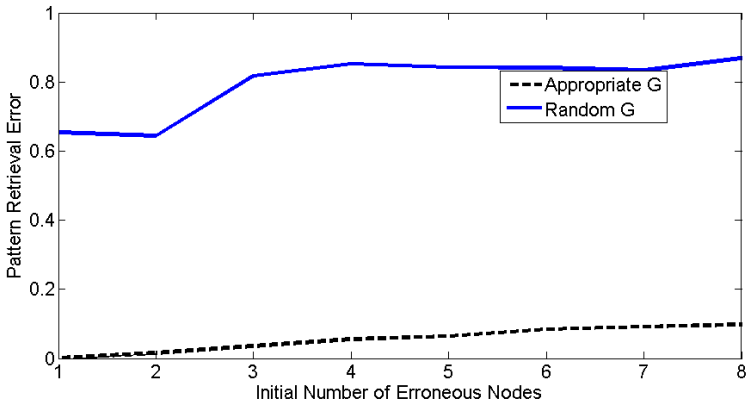

Figure 3: Pattern retrieval error rate against initial number of erroneous nodes with the amplitude of error varying between -2 and +2

assumed $u_{min} > 0$ which results in $y_{min} > 0$. We have to extend the results for $u_{min} = y_{min} = 0$ as well.

5 Conclusion

In this report, we have explained a lattice-based neural association mechanism that can memorize approximately an exponential number of associations. Simulation results show that with proper choice of weight matrices, we would be able to achieve a reasonably small pattern-retrieval error rate.

Our next step will be to properly design weight matrices according the second requirement mentioned in section 3, i.e. finding a matrix G such that itself and its pseudo-inverse both have a relatively small norm. Investigating the error performance in an analytical manner would be also subject to further research. Simulating the results for bigger networks is another topic which is to be addressed in future.