An Internal Mechanism for Detecting Parasite Attractors in a Hopfield Network

Jean-Dominique Gascuel
Bahram Moobed
Michel Weinfeld

Laboratoire d’Informatique de l’Ecole Polytechnique,
CNRS URA 1439, F-91128 Palaiseau Cedex, France

This paper presents a built-in mechanism for automatic detection of prototypes (as opposed to parasite attractors) in a Hopfield network. It has a good statistical performance and avoids host computer overhead for this purpose. This mechanism is based on an internal coding of the prototypes during learning, using cyclic redundancy codes, and leads to an efficient implementation in VLSI. The immediate separation of prototypes from parasite attractors can be used to enhance the autonomy of the networks, aiming at hierarchical multinetwork structures. As an example, the use of such an architecture for the classification of handwritten digits is described.

1 Introduction

Many biological and psychological facts show that natural information processing is performed in a very large number of functional units, each highly specialized in some particular simple task, connected together to form higher-level processing systems. One emerging trend in neural network research is to break down big networks into independent subunits, to implement many lower level and concurrent networks, each having a rather high connectivity but more loosely connected among them to form some kind of hierarchical network (Weinfeld 1990; de Bollivier et al. 1991).

Among various candidates for the basic building block of these assemblies, totally connected or Hopfield neural networks (Hopfield 1982) are very interesting because, in addition to their intrinsic associative properties, they are simple, have an architecture not depending on the problem to solve (except for their size), and can be integrated rather easily in silicon. In these networks, the learning process consists of encoding some chosen stable states, called prototypes, into the synaptic weights in a distributed fashion. This process introduces some spurious stable states, called parasites, in the network. To know that the network has converged toward a parasite can be useful: it may mean that the stimulus was
far enough from all the prototypes to be considered as not recognized, or rejected.

The focus of this paper lies on the associative memory applications of Hopfield neural networks. The state of such a network is defined by a "state vector" \( \sigma = (\sigma_i)_{i=1,\ldots,n} \), where \( \sigma_i \in \{-1, +1\} \) is the binary value associated with the \( i \)th neuron's activation. The popular Hebb learning rule has the disadvantage of rather low capacity, which is lower if the learned patterns are even moderately correlated (Hopfield 1982). On the contrary, the Widrow–Hoff learning rule enables the weight matrix \( W \) to be built iteratively and has been proven to converge to the so-called pseudoinverse rule (projection rule), with better capacity and tolerance to patterns' correlation (Personnaz et al. 1986; Diederich and Opper 1987).

We intend to build multinetwork architectures using basic associative VLSI components that we have designed (Johannet et al. 1992), to avoid the strong overhead of software simulations. Hence, it is necessary to give these components as much autonomy as possible. The first requirement is to include the learning mechanism in the chip itself. The second, at least as important, is to identify a stable state as a prototype or a parasite, with as little overhead as possible.

Straightforward methods are usually used for parasite identification. The most obvious one is to compare the current network attractor with the prototypes: this causes little overhead in a software simulation, but would have induced hardware and software overhead in the VLSI. Another method is to compute the energy of the attractor, knowing that the parasites have generally higher energy levels than the prototypes (Personnaz et al. 1986). But the architecture of our chip is such that the neuron's potentials are computed locally in each of them, without external broadcasting. Therefore an energy computation would have required a substantial increase of the design complexity. We propose a new way to implement what we call Parasite Detection Mechanism (PDM). Our method requires the addition of a simple device that increases very slightly the size and the complexity of the network, with a good compromise between this increase of complexity and the statistical performance of the detection.

The rest of the paper develops as follows: Section 2 introduces the basic concepts of this technique. Simulation results are presented in Section 3. A hardware implementation of PDM is briefly described in Section 4. Section 5 explains how to use this mechanism to improve retrieval. Finally, Section 6 describes a multinetwork classifier for handwritten character recognition, composed of several Hopfield networks that use PDM.

2 Automatic Parasite Detection Mechanism

The basic idea of PDM is to store some redundant data in the network during the learning phase and to verify it after each convergence. Storing
redundant data in a Hopfield network has already been done, but with significantly different goals and strategies (Personnaz 1986). The important improvement of our proposal is to automatically generate and check this redundant data. More precisely, let \( \sigma \) be a prototype to be stored, and \( C() \) a so-called “code function.” We propose to store the tuple \( (\sigma, C(\sigma)) \) in the network. During retrievals, when the network converges to a tuple \( (a, b) \), \( a \) will be declared to be a genuine prototype if \( b = C(a) \). We can already notice that, if \( d \) bits are devoted to the code size, the probability that a parasite has the good code, and hence is recognized as a prototype, is approximately \( 2^{-d} \), corresponding to a random drawing of a \( d \)-bit code. Therefore, \( d \) must be large enough to make this probability as low as needed. On the other hand, free evolution in Hopfield networks can be viewed as a nearest prototype search; the more neurons are devoted to \( C(\sigma) \), the more the search can be disturbed by these nonsignificant bits. Furthermore, increasing the network size might become expensive. Therefore, a compromise has to be found on \( d \).

2.1 Choice of the Code Function. Ideally, the code function \( C \) should provide unrelated codes over different stimuli. In practice, the code function cannot be assured to be injective because \( d \) must be notably smaller than the size of the input vector (to conserve space). Hence, \( C() \) must have a behavior similar to an error detection code function, meaning that changes in the vector state \( \sigma \) must cause a major change in the \( C(\sigma) \) value. Moreover, somehow depending on the learning rule used, parasites are often formed of thresholded linear combinations of prototypes (Amit et al. 1985), so PDM should be checked against this property.

The polynomial remainder functions are good candidates. The idea, based on the well-known Cyclic Redundancy Codes (Peterson and Weldon 1961), is to associate a polynomial in \( \mathbb{Z}/2\mathbb{Z}[X] \) of degree at most \( N - 1 \) to each \( N \)-sized state vector. The \( \psi \) function maps a \(-1\) (respectively \(+1\)) state of the neuron \( i \) to \( x_i^{-1} \) (respectively \( 0 \)). We can define \( \psi \) by

\[
\forall v \in \{-1, +1\}^N : \quad \psi_N(v) = \sum_{i=1}^{N} \frac{1 - v_i}{2} x_i^{-1}
\]

One can note that \( \psi \) is a homomorphism from the state vector space \( \{-1, +1\}^N \) with the term to term multiply (noted \( \otimes \)) to the polynomial space \( (\mathbb{Z})_{N-1}[X] \) with the addition modulo 2 (noted \( + \)).

The code function \( C \), for some fixed polynomial \( Q[X] \) of degree \( d \), is defined as

\[
C(v) = \psi_d^{-1}(\psi_N(v) \mod Q[X])
\]

\( Q[X] \) can easily be chosen so that the mechanism rejects the prototype’s opposite (always a stable attractor in Hopfield nets), that is, such that \( C(v) \neq -C(-v) \). Let \((-1)\) be the \( N \)-sized vector of all \(-1\), and
Parasite Attractors in a Hopfield Network

\[ R[x] = \psi_N(-1) = \sum_{i=0}^{N-1} X_i. \]

We get

\[
\psi(-v) = \psi((-1) \odot v) = R[X] \oplus \psi(v)
\]

and

\[
C(-v) = \psi_d^{-1}(R[X] \mod Q[X]) \odot C(v)
\]

We just have to choose some \( Q[X] \) such that \((R[X] \mod Q[X]) \neq \sum_{i=0}^{d-1} X_i \).

As \( \psi_N \) is a homomorphism, we have: \( C(a \odot b) = C(a) \odot C(b) \). So our mechanism cannot distinguish genuine prototypes from products of prototypes (with the \( \odot \) product). However, there is no direct relation between the \( \odot \) combinations and the thresholded linear combinations of prototypes. We have made simulations establishing that none (or very few) of the multiplications of prototypes is a stable attractor provided that we are working with a prototype set allowing the iterated Widrow-Hoff learning rule to work, that is, not too many prototypes, not too strongly correlated. The following results confirm this assertion.

3 Statistical Results

The success rate of PDM has been assessed by computer simulations on a Connection Machine 2 and on an Alliant FX-2800. Results are obtained by the following modus operandi: A set of \( p \) prototypes \( \pi_1 \cdots \pi_p \) is chosen at random and learned by the network, with the Widrow-Hoff learning rule iterated to approximate the pseudoinverse coefficients. Subsequently, we select a random vector \( \sigma \) at a known Hamming distance \( D_i \) from the nearest prototype \( \pi \). The network relaxes from this stimulus \((\sigma, C(\sigma))\) up to an attractor \((a, b)\).

- If \( a = \pi \), the retrieval is said to be successful: this is a "user" point of view.
- If \( C(a) = b \), the attractor is said to be accepted: this is the output of PDM.
- If both conditions do not agree, an error is reported: this is either an erroneous rejection or an erroneous acceptation.

Several simulations have been performed to explore the success, acceptance, and error rates, for different values of \( N, p, d \) and different ways of choosing a prototype set. Figure 1 shows typical data, obtained from the simulations. The results are displayed as follows: \( D_i \) is the normalized distance of the stimulus to the nearest prototype, Success is the rate at which the network relaxes to that prototype, and Accept is the rate at which the mechanism accepts the attractor as a prototype. The
Figure 1: Typical success curve, for \( N = 100, p = 20, \) and \( d = 5.\)

curve is the average of the relaxations obtained from 16 randomly chosen prototype sets.

The total success rate of one prototype set learned by the network can be assessed by the integral of this curve. Figure 2 shows the total success rates for several values of the load factor \( \alpha = p/N \) of the network (each point is the average over several prototype sets).

The aspect of this diagram can be viewed as a good semiquantitative characterization of the retrieval behavior of a Hopfield network in normal operation. We will use it in what follows to check possible degradation of the network properties due to the introduction of the code. Figure 3 shows the total retrieval success, without PDM \( (\circ, N = 100), \) with PDM \( (\circ, N = 100, d = 8), \) and with PDM and a reduced synaptic matrix\(^1\) \( (\triangle, N = 100, d = 8, \) and a matrix of size \( 108 \times 100).\) One can see that the mechanism does not seem to cause any significant changes on the retrieval performances since the various results lie approximately on the same curve. This result is important because it means that adding a mechanism such as PDM to an existing associative network does not change its retrieval performance.

Figure 4 shows the average identification error measured over several simulations, for values of \( d \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}, N = 100, \alpha = 0.3.\) It is difficult to carry out simulations much further because the meaning-

---

\(^1\) We call “reduced matrix” the synaptic matrix with the \( d \) last columns zeroed, after each learning step. This means that there is no influence of the code on the data parts of the full prototypes or stimuli.
Parasite Attractors in a Hopfield Network

Figure 2: Total success rates versus the load factor $\alpha$.

Figure 3: Comparing the total success rates: (○) without code, (○) with a code of size $d = 8$, (△) with a code of size $d = 8$, and a reduced matrix.

ful events (i.e., the errors of the mechanism) tend to become too sparse. The diagrams show that the maximum error observed decreases like $2^{-d}$. These results show that PDM effectively makes distinction between prototypes and parasites, that one can tailor the size of the system to the
quality needed by an application and that there is no significant effect on the basic associative properties of the Hopfield network.

4 The VLSI Implementation

We have implemented the Hopfield network with PDM in a VLSI chip, for the first time in 1989. In Gascuel et al. (1991) and Gascuel (1991), we describe this implementation (CMOS 1.2 μm, 1 cm², 420,000 transistors) that holds 64 binary neurons, 4096 synaptic weights, the relaxation, and the iterated Widrow-Hoff learning rule. We have chosen \( Q[X] = X^6 + X + 1 \), leading to a 6 bit code and 58 data neurons when PDM is activated (so the erroneous identification or rejection rate is less than \( 2^{-6} \), or \( \approx 2\% \)).

A systolic linear ring (Kung and Hwang 1988; Weinfeld 1989) is used to connect the neurons, each one having an arithmetic unit (12 bits wide), 64 weights (9 bits wide), and some convergence detection glue. With a cycle time of 80 nsec, a typical prototype set (15 moderately correlated vectors) is learned in \( \approx 1 \) msec (assuming off-chip prototypes arrive without wait state), and one retrieval (3 iterations on average) takes \( \approx 20 \) μsec.

On this chip, neural states \(+1\) and \(-1\) are coded by the sign bit (0 for \(+1\), and 1 for \(-1\)), directly using the mapping function \( \psi \) used for \( C() \). Polynomial remainders are easily implemented in hardware using “linear feedback shift registers” (Peterson and Weldon 1961). These intrinsically serial structures match very well the systolic loop semiparallel architecture of the network.
5 Using PDM to Improve Retrieval Success Rate

A possible strategy to increase the retrieval success rate is to add a random offset to the potential of each neuron in the network. This has been shown to improve the performances of the network (Peretto and Niez 1986; Amit et al. 1985). But it is necessary to compute a pseudorandom number for each neuron and at each step in the relaxation process, which would have made chip design much more complex. The ability to detect parasites has led to a retry strategy when a relaxation pass leads to a parasite. As we have already mentioned, a Hopfield network always finds an attractor (provided that a correct learning rule is used). When a parasite is detected, it is possible to restart a relaxation with a slightly different stimulus.

This may be viewed as a random search in the neighborhood of the input stimulus. It succeeds if there is an attractor basin close enough. The search field should not be made too large: first, because a larger neighborhood takes more time to have a reasonable coverage, and second, because a large neighborhood may contain patterns leading to other prototypes. This strategy using PDM offers good performance and needs few random bits. Figure 5 shows the success rates obtained with such a strategy when allowing no retry or one to five retries. The gain can reach up to 25%. This result shows that PDM may indeed be used to significantly improve the retrieval success rate of a Hopfield neural network. The overall cost of this strategy is low: a small CRC generator for PDM and a low number of retries are sufficient. Furthermore, the computations used have simple and fast implementations both in hardware and software.

6 An Application of PDM: Handwritten Digit Recognition

We propose a multinetwork architecture for classification, composed of binary Hopfield networks using PDM. In this approach, the number of Hopfield networks used is equal to the number of classes of the problem ($N_\omega$); there is a Hopfield network $H_i$ dedicated to each class $\omega_i$. To classify an input pattern $\sigma$, it is fed to all the $N_\omega$ networks, and each delivers a Boolean response signal $d_i$ to indicate if it considers $\sigma$ as a member of its goal class $\omega_i$ or not. For this purpose, each network $H_i$ learns some prototype set $\Pi_i$ composed of some representative patterns of its goal class $\omega_i$. Let $\sigma'$ be the output attractor when $\sigma$ is presented to $H_i$. If we suppose that convergence toward one of the prototypes in $\Pi_i$ is a good indication that the pattern belongs to the class $\omega_i$, that is to say

$$\sigma' \in \Pi_i \Leftrightarrow \sigma \in \omega_i$$  \hspace{1cm} (6.1)
then the desired output $d_i$ can be defined as the Boolean signal $d_i = (\sigma' \in \Pi_i)$. PDM is the key feature that allows us to generate efficiently this signal $d_i$, with a low error rate.

Some ambiguous input patterns may be recognized by more than one network. In this case, an algorithm called the arbiter is needed to take an appropriate action: either determine the class of the pattern or start another process that gives more information about the competing classes.

6.1 Artificial Parasites. Learning only some prototypes of the goal class $\omega_i$ causes some problems:

- Usually, the prototypes belonging to the same class are correlated, and it is known that correlated prototypes decrease the performance of the Hopfield network (Personnaz et al. 1986), even when the pseudoinverse rule is used.

- As each Hopfield network is responsible only for its goal class, only the patterns of the goal class should converge toward one of the prototypes (relation 6.1). Therefore the patterns of the other classes should converge to some parasite attractors. As there is no direct control on the number and location of the parasites created during learning, their attraction basins may cover only partially the space of the other classes.
The tests realized with different prototype sets show that these problems cut down the performance of the networks. To overcome this problem, it is necessary to create some other parasite attractors that can attract the patterns of the other classes. We propose to train each network $H_i$ with some pattern set $\Theta_i$, composed of some patterns not belonging to the goal class, in addition to the prototypes in $\Pi_i$. These new attractors should not activate the recognition signal $d_i$ and must be detected as parasites. As a CRC code is used in the parasite detection mechanism, the easiest way to differentiate these attractors from prototypes is to append a false CRC code to these attractors (called artificial parasites hereafter) during learning.

### 6.2 Prototypes and Artificial Parasites Selection

The performance of this classifier depends strongly on the prototypes and artificial parasites chosen for each network. In fact, the prototype set $\Pi_i$ must roughly cover the space of the goal class, and the artificial parasite set $\Theta_i$ must cover the space of all the other classes, as well as possible. As this is essentially a vector quantization task, we employ an LVQ network (Kohonen 1988) to determine the $\Pi_i$ and $\Theta_i$ sets. After choosing $N_{\Pi_i}$ (number of prototypes in $\Pi_i$) and $N_{\Theta_i}$ (number of artificial parasites in $\Theta_i$), an LVQ network with $N_{\Pi_i} + N_{\Theta_i}$ neurons is trained with the patterns of the learning database. We have tested the two following strategies for choosing $\Pi_i$ and $\Theta_i$ for a network $H_i$:

1. Using LVQ with $N_{\omega}$ classes. The $N_{\Pi_i}$ prototypes are taken from the neurons corresponding to class $\omega_i$, and the $N_{\Theta_i}$ artificial parasites are taken from the neurons corresponding to all the other classes.

2. Using LVQ with only two classes: the goal class $\omega_i$ and a class containing all the other patterns. Here, the $N_{\Theta_i}$ artificial parasites are taken from the neurons corresponding to the second class.

After this procedure, the $N_{\Pi_i} + N_{\Theta_i}$ patterns given by LVQ are binarized and learned by the Hopfield network $H_i$, and the process is repeated for all networks $H_i$. The necessary binarization of the LVQ reference vectors may cause some loss of performance. The task of the artificial parasites is to attract the patterns of all the other classes, but no distinction is needed among them. Therefore the second method, being less constrained, seems to be more adapted to this problem, which is confirmed by the tests. The choice of $N_{\Pi_i}$ and $N_{\Theta_i}$ is important. If they are chosen too small, there may not be enough attractors in the space of the classes. On the other hand, if they are chosen too large, the patterns can overload the Hopfield network and cut down its performance.

### 6.3 The Arbiter

When several networks activate simultaneously their recognition signals, an arbiter is needed to select a winner them. Two simple arbiters are:
Figure 6: Architecture of the multinetwork classifier used for handwritten digit recognition.

- The network that gives the smallest Hamming distance between the input pattern and the converged output pattern is considered as the winner.

- Several slightly noised versions of the pattern, as well as the original one, are presented to the networks, and the network that recognizes most frequently wins. The noise added must not change too many bits, otherwise the disturbed pattern may become too different from the original to be recognized. This mechanism has been described in Section 5.

In the tests, a combination of these two arbiters is used. In other words, the network that has most frequently given a nearest output pattern wins. A more appropriate choice of distorted versions (such as slightly translated, thinned or rotated images) may produce better results.

6.4 Experiments. As an example, we have tested this classifier for handwritten digit recognition (Fig. 6). Efficient methods for handwritten
digit recognition have already been suggested (Le Cun et al. 1990; Simard et al. 1992); we do not intend to suggest a better method, but just show that ours gives fairly acceptable results with rather low overhead, when dealing with this classical problem. This makes us confident about the general approach that we use for implementing multinet systems based on our integrated network.

The database\(^2\) contained 8700 patterns, each one being a 16 × 16 binary image of a size normalized digit. The database was divided into a learning set of 4000 patterns, and a test set of 4700 patterns. The classifier included 10 Hopfield networks of 266 neurons (256 bits for image + 10 bits for CRC) trained by the Widrow-Hoff learning rule. The results given are all for the combined arbiter that gave the best results. With only 8 prototypes and 6 artificial parasites for each class, 92.1% of the characters were recognized correctly and 3.4% were rejected. With 16 prototypes and 14 artificial parasites, 93.1% of the characters were recognized correctly and 3.1% were rejected. The results are satisfying, taking into account the relatively low number of prototypes or artificial parasites stored in each network. The number of prototypes and artificial parasites cannot be increased too much because of the limited capacity of the Hopfield networks.

7 Conclusion

The Parasite Detection Mechanism presented in this paper is rather easy to control and cheap to implement in hardware. Furthermore, the information that it provides (convergence toward a prototype or a parasite) may be used to construct enhanced relaxation strategies, as shown in Section 5. This mechanism has also been the key feature in the use of Hopfield network in a multinet architecture. By using the artificial parasites and the LVQ network for the selection of the prototypes and the artificial parasites, this architecture has been applied to the recognition of handwritten digits.

This multinet architecture has the advantage of modularity: modifications or optimizations of a particular network, or in some cases addition of a new class, might be done separately without disturbing much the other networks. Other applications may also use PDM to trigger reprocessing or even learning in some other kind of multinet architectures. We believe that a mechanism such as PDM is an efficient way to help implementing autoadaptivity in neural chips and, further, in higher-level neural architectures.

\(^2\)Supplied by ESPCI (Knerr et al. 1992).
Acknowledgments

This work received support from DRET contract 87-187, the Groupement Circuit Intégrés Silicium, and the Projet de Recherche Coordonnées Architectures Nouvelles de Machines. We want to thank A. de Maricourt for the preliminary work done on this subject, D. Descourse and E. Lemoine for the simulations done on the Connection Machine, and L. Montoliu for simulations on the Alliant computer. We are grateful to the referees for their fruitful comments.

References

Parasite Attractors in a Hopfield Network


Received April 6, 1993; accepted December 15, 1993.