A new Algorithm for Implementing BSB-based Associative Memories

DANIELE CASALI1, GIOVANNI COSTANTINI1,2, RENZO PERFETTI3, ELISA RICCI3
1Department of Electronic Engineering
University of Rome “Tor Vergata”
Via del Politecnico, 1 – 00133 ROMA
ITALY
2Institute of acoustics “O. M. Corbino”
Via del Fosso del Cavaliere, 100 – 00133 ROMA
ITALY
3Department of Electronics and Information Engineering
University of Perugia
Via G. Duranti, 93 – 06125 PERUGIA
ITALY

Abstract: - The relation existing between support vector machines (SVMs) and recurrent associative memories is investigated. The design of associative memories based on the generalized brain-state-in-a-box (GBSB) neural model is formulated as a set of independent classification tasks, which can be efficiently solved by standard software packages for SVM learning. Some properties of the networks designed in this way are evidenced, like a surprising generalized Hebb’s law. The performance of the SVM approach is compared to existing methods with non-symmetric connections, by some design examples.

Key-Words: - Associative memories, neural networks, support vector machines.

1 Introduction

Since the seminal paper of Hopfield [1], a lot of synthesis methods have been proposed for the design of associative memories (AM) based on single-layer recurrent neural networks. The problem consists in the storage of a given set of prototype patterns as asymptotically stable states of a recurrent neural network [2]. A review of some methods for AM design can be found in [3]. In this paper we adopt the generalized brain-state-in-a-box (GBSB) neural model introduced by Hui and Zak [4]. This model was analyzed by several authors and its usefulness for AM design is known [5-7]. In particular we consider the design of GBSB neural networks with non-symmetric connection weights. This design problem was addressed in [5], where the weights are computed using the pseudo-inverse of a matrix built with the desired prototype patterns. Moreover, in [5] a method to determine the bias vector is suggested, in order to reduce the number of spurious stable states. In [6] the design of the AM is reformulated as a constrained optimization problem and an analog “designer” neural network is proposed to find the optimal solution at steady state.

In the past few years, SVMs aroused the interest of many researchers being an attractive alternative to multi-layer feed-forward neural networks for data classification and regression [8]. The basic formulation of SVM learning for classification consists in the minimum norm solution of a set of linear inequality constraints. This optimization problem can be easily adapted in order to incorporate the conditions for asymptotic stability of the states of a recurrent neural network. So, it seems useful to exploit the relation between these two paradigms in order to take advantage of some peculiar properties of SVMs: the “optimal” margin of separation, the robustness of the solution, the availability of efficient computational tools. In fact, the SVM learning problem has no non-global solutions and can be solved by standard routines for quadratic programming (QP); in the case of a large amount of data, some fast solvers for SVMs are available, e.g. SVMlight [9].

The paper is organized as follows. In Section II we briefly review SVMs with linear kernel function. In Section III we examine the relation existing between the design of a recurrent associative memory based on the GBSB model and support vector machines. In Section IV we compare the proposed approach to some existing techniques, through some design examples and simulation results. Finally, in Section V some conclusions are given.

2 Support Vector Machines

Let \((x_k, y_k), k = 1,\ldots, m\) represent the training examples for the classification problem; each example \(x_k \in \mathbb{R}^N\) belongs to the class \(y_k \in \{-1, +1\}\). Assuming linearly separable classes, there exists a separating hyper-plane such that

\[ y_k (w^T x_k + b) > 0 \quad k = 1,\ldots, m \]  

(1)

The minimum distance between the data points and the separating hyper-plane is the margin of separation. The goal of a SVM is to maximize this margin. We can rescale
Once this quadratic programming problem (QP) has been solved, we have the optimum Lagrange multipliers \( \alpha_1, \ldots, \alpha_m \). As a consequence, the margin of separation is \( 1/||w|| \) and maximization of the margin is equivalent to the minimization of the Euclidean norm of the weight vector \( w \).

The corresponding weights and bias represent the optimal separating hyper-plane (Fig. 1). The data points \( x^k \) for which the constraints (2) are satisfied with the equality sign are called support vectors. Introducing the Lagrange multipliers \( a_1 \ldots a_m \), the minimization of \( ||w||^2 \) under constraints (2) can be recast in the following dual form [7]:

\[
J(\alpha) = -\sum_{k=1}^{m} \alpha_k + \frac{1}{2} \sum_{k=1}^{m} \sum_{j=1}^{m} \alpha_k \alpha_j y_k y_j (x^k)^T x^j
\]

subject to the linear constraints

\[
\alpha_k \geq 0 \quad k = 1, \ldots, m
\]

(4)

Once this quadratic programming problem (QP) has been solved, we have the optimum Lagrange multipliers \( \alpha_k \), one for each data point. Using Lagrange multipliers we can compute the optimum weight vector \( w \) as follows

\[
w = \sum_{k=1}^{m} \alpha_k y_k x^k
\]

(6)

Only the Lagrange multipliers corresponding to support vectors are greater than zero, so the optimum weights depend uniquely from the support vectors. For a support vector \( x^k \) it is \( w^T x^k + \beta = y_k \) from which the optimum bias \( \beta \) can be computed. In practice it is better to average the values obtained by considering the set \( SV \) of all support vectors.

Using the optimal weight vector \( w \) we can compute the optimum weight vector \( w \) as follows

\[
J(\alpha) = -\sum_{k=1}^{m} \alpha_k + \frac{1}{2} \sum_{k=1}^{m} \sum_{j=1}^{m} \alpha_k \alpha_j y_k y_j (x^k)^T x^j
\]

subject to the linear constraints

\[
0 \leq \alpha_k \leq C \quad k = 1, \ldots, m
\]

(10)

In this case, for the optimum Lagrange multipliers we have: \( 0 < \alpha_k < C \) for support vectors \( (\xi_i = 0) \); \( \alpha_k = 0 \) for points correctly classified with \( \xi_i < 0 \); \( \alpha_k = C \) for points with \( \xi_i > 0 \) (are misclassified if \( \xi_i > 1 \)).

Finally, the threshold \( \beta \) could be chosen a priori; in this case, assuming linearly separable classes, the SVM can be trained by finding the minimum of

\[
J(\alpha) = -\sum_{k=1}^{m} (1 - \beta) \alpha_k + \frac{1}{2} \sum_{k=1}^{m} \sum_{j=1}^{m} \alpha_k \alpha_j y_k y_j (x^k)^T x^j
\]

subject to the bound constraints

\[
0 \leq \alpha_k \leq C, \quad k = 1, \ldots, m
\]

With respect to the general formulation, the equality constraint (4) disappears and it is easier to find the solution. From the optimum Lagrange multipliers we obtain the weight vector \( w \) using (6).

3 SVM Design of GBSB Neural Networks

Let consider the Generalized Brain-State-in-a-Box (GBSB) neural model described by the following difference equation:

\[
x(t+1) = g[x(t) + \alpha(w x(t) + b)] \\
\]

\[
t = 0, 1, 2, \ldots, (12)
\]

\[
x(t) = [x_1(t), \ldots, x_d(t)] \in [-1,+1]^n \quad \text{is the state vector; } n \text{ is the number of neurons; } W \in \Re^{m \times n} \text{ is the weight matrix assumed non symmetric; } \beta \in \Re^n \text{ is the bias vector; } \alpha \text{ is a scalar; } g \text{ is a vector valued function, whose } i\text{th component is defined as}
\]
associative memory by solving (13) with respect to the
for a given binary equilibrium point
each example

\[
\begin{align*}
(g(y))_i &= 1 \quad \text{if} \quad y_i \geq 1 \\
(g(y))_i &= y_i \quad \text{if} \quad -1 < y_i < 1 \\
(g(y))_i &= -1 \quad \text{if} \quad y_i \leq -1.
\end{align*}
\]

Let \( B = [-1, +1] \). The design of a bipolar associative
memory, based on model (12), can be formulated as follows.

Find the connection matrix \( W \) and bias vector \( b \) such that:

\( \cdot \) a given set of bipolar vectors \( x_1 \ldots x^m \in B^n \) represent as
many asymptotically stable equilibria of system (12);

\( \cdot \) the attractivity of the desired equilibria is as large as
possible;

\( \cdot \) the number of not desired (spurious) stable equilibria and
the number of oscillatory solutions are as small as possible.

Several analysis results are available for GBSB neural
networks. In the following we review only the most
significant of them (the proofs can be found in [6]):

\( a. \) If \( w_{ii} \geq 0 \) for \( i = 1, \ldots, n \), only the vertices of \([-1, +1]^n\)
can be asymptotically stable equilibria of system (12).

\( b. \) \( x \in B^n \) is an asymptotically stable equilibrium point of
system (12) if and only if

\[
\sum_{j=1}^{n} w_{ij} x_i x_j > 0 \quad i = 1, \ldots, n \tag{13}
\]

Constraints (13) represent existence and stability conditions
for a given binary equilibrium point \( x \). We can design the
associative memory by solving (13) with respect to the
weights and bias for a given set of bipolar equilibrium
points \( x^1 \ldots x^m \).

c. Assume that \( x \in B^n \) is an asymptotically stable equilibrium
point of system (12). Then, no equilibrium point exists at
Hamming distance one from \( x \) if \( w_{ii} = 0 \), for \( i = 1, \ldots, n \).

To summarize, a zero-diagonal connection matrix guarantees
the absence of equilibria with non-binary components,
as well as the absence of equilibria at Hamming distance one from the desired patterns. Taking
into account the stability conditions (13) and the zero-
diagonal constraint, the synthesis problem for the ith neuron
can be recast as follows: find \( w_i \) and \( b_i \), \( i = 1, \ldots, n \) such that

\[
x^k_i (w_i x_i^k + b_i) > 0 \quad k = 1, \ldots, m \tag{14}
\]

where \( x_i^k \in B^{n-1} \) is equal to \( x_i \) with the ith entry eliminated and \( w_i \in R^{n-1} \) is the ith row of matrix \( W \) with the ith entry
eliminated. Comparing expressions (1) and (14), the design
of the ith neuron can be interpreted as a classification
problem with training set given by \((x_i^k, x_i^h), k = 1, \ldots, m;\)
each example \( x_i^k \) is void of the component \( x_i^k \) and belongs
to a class represented by \( x_i^h \in \{-1, +1\} \). The synthesis
problem can be solved by a set of \( n \) independent SVMs,
where SVM(i), \( i = 1, \ldots, n \), computes the weights \( w_i \) and bias \( b_i \) for ith neuron. A similar viewpoint was adopted in [9] to
design a continuous-time AM by \( n \) perceptrons trained by
Rosenblatt’s algorithm. Using SVMs we have a
maximization of the margin of separation.

Some useful relations can be found between the properties
of SVMs and the design of model (12).

**Property 1.** Given \( n \) arbitrary vectors in \( B^n \), they can be
stored as asymptotically stable states of (12), with a zero-
diagonal \( W \), provided that they differ in at least two components.

**Proof.** The Vapnik-Chervonenkis (VC) dimension of
oriented hyper-planes in \( R^n \) is \( n+1 \) [7]. That is, given \( n+1 \)
arbitrary distinct points in \( R^n \), with arbitrary class labeling,
they can be separated by an oriented hyper-plane. In the
present case, each SVM handles vectors in \( B^{n-1} \), so \( n \) of
them can be arbitrarily shattered. The only exception is
when two prototypes \( x_i^k \), \( x_i^j \) differ in only one component, say the \( i \)th; in this case, the corresponding reduced vectors
\( x_i^k \), \( x_i^j \) used to train the \( i \)th machine will be identical but
belonging to different classes.

**Property 2.** The weights of the GBSB neural network
computed using SVMs follow a generalized Hebb’s law.

**Proof.** Using (6) with \( y_i = x_i^k \), we have the following
expression for the weights

\[
w_{ij} = \sum_{k=1}^{m} \alpha_i^k x_i^k x_j^k \quad i \neq j \tag{15}
\]

where \( \alpha_i^k \) is the Lagrange multiplier corresponding to the ith neuron and the \( k \)th prototype vector. Expression (15)
corresponds to the Hebb’s law where each product \( x_i^k x_j^k \) is
multiplied by a real constant \( \alpha_i^k \geq 0 \). Note that a subset of
the Lagrange multipliers will be zero (corresponding to non-
support vectors), hence not all the products \( x_i^k x_j^k \) will be
present in expression (15).

**Property 3.** Using the SVMs to design a GBSB neural
network we always obtain a unitary margin of stability, i.e.
\[ \min x_i^k (w_i^T x_i^k + b_i) = 1 \quad i = 1, \ldots, n \]

**Remark 1.** Using Property 3 we can easily take into account the effects of parameter variations. Assume the weight vector \( w_i \) is affected by a variation \( \Delta w_i \); in order to preserve the stability of \( x_i^k \) condition (14) becomes

\[ x_i^k (w_i^T x_i^k + b_i) > x_i^k \Delta w_i^T x_i^k \]

i.e.

\[ x_i^k (w_i^T x_i^k + b_i) = -x_i^k \Delta w_i^T x_i^k \]

Using the worst case approach and taking into account Property 3 we have

\[ (n-1) |\Delta w|_{\text{max}} < 1. \]

Assume that the weights are represented in digital form with \( B+1 \) bits; the previous condition becomes

\[ (n-1) 2^{(B+1)-1} < 1 \]

The required number of bits by means of which the weights is easily obtained:

\[ B+1 > \log_2(n-1) \]

It is worth noting that this number only depends on the memory size and not on the stored patterns.

**Remark 2.** A second application of Property 3 is a preselection of the bias \( b_i \). Using the SVMs, as explained above, some of the negatives of the stored patterns \( -x_1, \ldots, -x_m \) could be stable states (attractors) of the AM. To avoid this problem, the values \( b_i \) can be computed as follows. The vector \( -x_k \) cannot be a stable state, if \( -x_k^T (-w_i^T x_i^k + b_i) < 0 \), for at least one neuron \( i \). Assume that the prototype \( x_i \) is a support vector for the \( i \)th SVM; hence

\[ x_i^k (-w_i^T x_i^k + b_i) = 1 \quad (16) \]

Replacing \( x_i \) with \( -x_k \) in (16) gives

\[ -x_k^T (-w_i^T x_k^k + b_i) = x_i^T (w_i^T x_i^k + b_i) - 2x_i^k b_i = 1 - 2x_i^k b_i \]

To make \( x_k \) unstable it is sufficient that \( b_i > 1/2 \) if \( x_i^k = +1 \) and \( b_i < -1/2 \) if \( x_i^k = -1 \).

As evidenced by computer experiments, if we use \( n \) SVMs to store \( m \leq n \) vectors, each \( x_k \) will be a support vector for most of the \( n \) machines; so we suggest the following expression

\[ b_i = (0.5 + \varepsilon) \left( \frac{1}{m} \sum_{k=1}^{m} x_i^k \right) \quad (17) \]

where \( \varepsilon \) is a small positive number. The sign of \( b_i \) is selected according to the majority of +1 or −1 in position \( i \); in this way we maximize the number of negative prototypes being unstable for neuron \( i \). Note that expression (17) is in accordance with the method suggested in [5,7] to bias the trajectory toward one of the desired patterns.

**Remark 3.** No convergence results are known for model (12) with non-symmetric matrix \( W \). However, the simulation results show that there is a strict relation between the value of \( \alpha \) in (12) and the magnitude of the weights. If we increase the magnitude of the weights, a correspondingly lower value of \( \alpha \) should be used in order to have a convergent dynamics for most initial conditions. This behavior is well known for symmetric weight matrices;

\[ \text{in this case convergence is guaranteed if } \alpha < 2/|\lambda_{\text{min}}(W)| \]

where \( \lambda_{\text{min}}(W) \) represents the minimum real eigenvalue of \( W \); since the diagonal elements of \( W \) are zero, \( \lambda_{\text{min}} \) is negative and its magnitude increases with the norm of \( W \). Using SVMs we obtain the minimum norm weight vector \( w_i \).

**4 Experimental Results**

In this Section we present a design example and simulation results, in order to compare the SVM design with different existing methods, giving non-symmetric \( W \). SVM training was carried out using SVMlight [9]. The simulations was carried out using \( \alpha = 0.3 \) in eqn.(12).

In this example, we compare the SVM solution with the design obtained by a perceptron trained by Rosenblatt’s algorithm [10]. Using this algorithm, we must select a learning rate \( \eta \) which affects the performance of the designed AM. We tried different values for \( \eta \); the margins of separation are comparable for \( 0.5 \leq \eta \leq 1 \); for \( \eta < 0.5 \) the margins can be very small for some patterns; so, in the following, we present only the results obtained with \( \eta = 1 \).

This choice presents the additional advantage of integer connection weights [10].

For this test, ten training sets were randomly generated; each set is composed of \( m = 5 \) patterns with size \( n = 10 \) \((m = 0.5n)\). The minimum Hamming distance between patterns is two. For each set we found the weight matrix and bias vector using the SVMs and the perceptron learning algorithm. Both methods always stored all the five vectors as asymptotically stable states. The average number of spurious stable states was 4 with the SVMs and 5.7 with the perceptron learning algorithm. Then we simulated the dynamic behavior of the GBSB network, starting from initial states with Hamming distance \( H \) from each of the stored prototype vectors. The results are summarized in Table I; they are based on ensemble averages over ten

| TABLE I – Example 1: convergence test for 1024 initial states |
|-----------------|-------|-------|-------|-------|
| convergence to   | SVM   | perceptron |
| closest stored  | 809   | 556   |
| pattern         | 64    | 173   |
| stored pattern  | 144   | 250   |
| (not the closest)| 7     | 45    |

| TABLE II – Example 1: convergence test for 1024 initial states |
|-----------------|-------|-------|-------|-------|
| convergence to   | SVM   | perceptron |
| closest stored  | 809   | 556   |
| pattern         | 64    | 173   |
| stored pattern  | 144   | 250   |
| (not the closest)| 7     | 45    |
training sets. The entry corresponding to $H = p$ represents the average number of initial states at Hamming distance $p$ from one of the stored patterns $x^k$, converging to $x^k$ itself. To gain a deeper insight into the behavior of the designed associative memory, we simulated its dynamic evolution, starting from all the 1024 bipolar initial states. Then, we counted the number of initial states converging to the closest stored pattern (in Hamming sense), the number of initial states converging to one of the stored patterns but not to the closest one, the number of initial states converging to binary spurious states and, finally, those corresponding to oscillatory solutions (convergence is not guaranteed, since $W$ is non-symmetric). These results are shown in Table II. By using SVMs, we obtain less trajectories converging to spurious stable states or not converging.

We examined also the capacity measure proposed in [11], based on the concepts of recall accuracy (RA) for a given Hamming radius $\rho n$ $(0 < \rho < 1)$. For example, $H=1$ in Table I corresponds to $\rho = 0.1$; since in $B^{10}$ there are ten vectors with $H=1$ from a given prototype, the recall accuracy (RA) is 99% with the SVM and 84% with the perceptron. The value $H=2$ corresponds to $\rho = 0.2$; taking into account that there are 45 vectors at $H = 2$; the recall accuracy (RA) is 86% with the SVM and 58% with the perceptron. For comparison purposes, we consider the results in [11], based on the Ho-Kashyap method [12]: they are RA>95% for $\rho=0.1$ and $m < 0.3 \ n$; RA>80% for $\rho=0.2$ and $m < 0.25 \ n$.

4 Conclusion

We showed how the design of a neural associative memory can be recast as a classification problem and solved by one of the available packages for SVM training. The proposed approach presents some interesting features:

- nonbinary stable states are avoided;
- $m \leq n$ arbitrary vectors with $n$ components can always be stored as asymptotically stable states, provided that they differ in at least two positions;
- the storage of the negatives of the desired patterns can be avoided in a simple way;
- the attractivity of the stored patterns compares well with existing techniques;
- the minimum margin of stability is normalized to one for every set of prototypes and memory size;
- the minimization of weight vector norm helps to obtain a convergent behaviour;
- computationally efficient methods are available to solve the design problem;
- no design parameters must be chosen a priori or empirically tuned provided that $m \leq n$;
- given $m > n$ patterns, different subsets of them can be stored, trading off capacity and error correction ability by adjusting a design parameter.

References: