Training and Holistic Computation of Vector Graphics with Hebbian Bases in Contrast to RAAM Networks

Mark Schaefer and Werner Dilger
Chemnitz University of Technology
Chemnitz, Germany
Email: mark@markschaefer.de, dilger@informatik.tu-chemnitz.de

Abstract—Hebbian Learning is well-known for training of associative networks whereas RAAM learning uses auto-associative networks which are trained to represent structured information like parse trees of natural sentences or logical terms.

In this paper Hebbian learning is used for representing structured information in terms of vector graphic. The resulting networks are holistically computed. Furthermore a theorem relating bipolar Hebbian learning is proved.

I. INTRODUCTION

Recursive auto-associative memory (RAAM) networks were introduced by J.B. Pollack [5] for the purpose of learning representations of structured objects that do not increase with the size of the structures represented. RAAM networks were typically applied to syntactical structures like parse trees of natural language sentences [6] or logical expressions (cf. [4]). Hebbian learning was described 1949 by Donald Hebb as a learning rule for the weights in connectionist networks [3]. It can be used for one-step training of associative networks.

In this paper we will show that Hebbian learning is as suitable for representing structured information as RAAM networks are and that the learned information can be holistically modified.

In section II RAAM networks and their holistic modification are described, followed by a section which deals with the communities and differences of RAAM and Hebbian learning. In section III Hebbian bases are defined and a relating theorem is given and proved. The next section handles the holistic modification of matrices resulting from Hebbian learning and the results of concrete examples.

II. RAAM NETWORKS AND HOLISTIC MODIFICATION

A. Structure

Recursive auto-associative memory networks (RAAM) were developed by Pollack [5]. Figure 1 shows their basic structure. A RAAM is a three-layer feed-forward network. The input and output layer consist of k slots, the middle layer of only one. Each slot has n positions (or bits) and is independent from the other ones in the same layer. For short this is denoted as n-k-RAAM network or just n-k-RAAM.

The input neurons together with the hidden neurons are called encoder and the hidden neurons and the output neurons are called decoder. A RAAM is trained as an auto-associator, i.e. the network should reproduce each input vector at the output layer. When a training example has been learned successfully one can use the activation of the hidden neurons as a representation of the input/output vector. The input vector is encoded by the first layer of weights and the representation is decoded by the second one.

B. Structured Representation

After one knows a representation of an input vector it can be used as a part of another input vector, which is encoded into its own representation and so on. This is the essential feature of RAAMs. It can be used for subsymbolic representation of trees and their holistic computation, i.e. the computation of the whole RAAM network at once in order to gain another RAAM network which represents different information. It is important to bear in mind that the representation of every node changes during each training cycle which effects dependent nodes.

Once a RAAM has learned a tree, only the decoder part is needed to unfold it completely. Starting with the representation of the topmost node - which is now used as an input for the decoder network - all nodes are reconstructed.

C. Holistic Computation

If a RAAM network was trained to represent a tree, it can be computed holistically. There are three types of holistic computation: transformation, classification and matching. They have in common that

... holistic inference maps directly from the representation of a problem to the representation of an answer, in a gestalt fashion without accessing the constituent elements or relations within the data. [2]
For example holistic transformation was done by Chalmers who trained RAAM networks to represent 40 English sentences and their corresponding passive form [1]. The half of the active-passive pairs (in form of the representation of the topmost node) were used as training patterns for a so-called transformation network. After training it was able to generate the passive form of 65% of the remaining sentences.

III. HEBBIAN LEARNING IN CONTRAST TO RAAM LEARNING

If a given tree is learned by a RAAM network the encoder part is not needed any longer. The remaining decoder part is an n-kan associative network. Such a network representing structured information can be built from scratch by using Hebbian learning.

A. Principles of Hebbian Learning

Donald Hebb [3] specified the following learning rule for the weight $w_{xy}$ of a connection between two neurons $x, y$ in a neural network.

$$\Delta w_{xy} = \eta \cdot x \cdot y$$

For an associative network and a training pattern $(x_i, y_i)$

$$x_i = (x_1^i, x_2^i, \ldots, x_n^i)$$
$$y_i = (y_1^i, y_2^i, \ldots, y_n^i)$$

we gain the following so called correlation matrix which is the outer product of the input and output vector:

$$W_i := x_i^T y_i$$

All correlation matrices are summed up to the final weight matrix of the associative network:

$$W = W_1 + W_2 + \ldots + W_k$$

The associated output vector $y_i$ is retrieved by multiplying $x_i$ with $W$:

$$x_i W = x_i W_1 + x_i \sum_{j=1 \ldots k} W_j$$
$$= x_i (x_i^T y_i) + \sum_{j=1 \ldots k, j \neq i} x_i (x_j^T y_j)$$
$$= (x_i)^2 y_i + \sum_{j \neq i} (x_i x_j^T) y_j$$

We gain $y_i$ multiplied with a positive constant plus a term called crosstalk. Obviously the crosstalk equals zero if $x_1, x_2, \ldots, x_k$ are piecewise orthogonal.

B. Learning of Structured Information with Hebbian Bases

Any information given by a tree can be trained using Hebbian learning similar to the process of RAAM learning described in section II. For training a tree a set of orthogonal vectors $X = \{x_1, x_2, \ldots, x_k\}$ is needed. Each node of the tree is represented by a unique vector $x_i \in X$ and for each of them a training pattern is built, consisting of the $x_i$ as input and the representations of the children $c_{i1}, c_{i2}, \ldots, c_{ik}$ as output. The correlation matrix is built by multiplying $x_i$ with the concatenated representations of the child nodes.

$$W_i = x_i^T \cdot (c_{i1} | c_{i2} | \ldots | c_{ik})$$

This procedure is done for every node in the tree and the correlation matrices are added. Of course the representation of a node $x \in X$ can be used on the right side of (9) in order to represent a tree. This method of splitting a tree into training patterns is analogue to the one described for RAAM networks. In contrast to RAAM training the representations of the nodes are given from the beginning and only one training step must be performed. Furthermore no particular order for their training is prescribed.

Since the right signs of the associated vectors but not the vectors itself are returned, a binary bipolar encoding of the patterns is useful. Therefore a set of bipolar orthogonal vectors is needed, which is the subject of the next section.

C. Hebbian Bases

This subsection deals with orthogonal bipolar vectors. An algorithm for generating sets of orthogonal bipolar vectors called Hebbian bases is specified and the following theorem is proved: Hebbian bases only exist in dimensions $n = 2^k$. To do so we start with some basic definitions and lemmas.

1) Fundamentals:

Definition 3.1: Set $H := \{-1, +1\}$. A vector $b \in H^n$ is called bipolar. The $j$-th position of $b$ is denoted as $b_j$. A set $B_n = \{b_1, b_2, \ldots, b_n\}$ of bipolar vectors $b_j \in H^n$ is called Hebbian base if $\langle b_i, b_j \rangle = 0 \forall 1 \leq i < j \leq n$.

Definition 3.2: The hamming difference of two bipolar vectors $b_1, b_2 \in H^n$ is defined as:

$$hdf(b_1, b_2) := \sum_{i} |b_1^i \neq b_2^i|$$

Lemma 3.3: Two bipolar vectors $b_1, b_2 \in H^n$ are orthogonal if and only if $hdf(b_1, b_2) = n/2$.

Proof:

$$b_1, b_2 \text{ are orthogonal} \iff \langle b_1, b_2 \rangle = 0$$
$$\iff 0 = \sum_{b_1^i = b_2^i} b_1^i b_2^i + \sum_{b_1^i \neq b_2^i} b_1^i b_2^i = \sum_{i=1}^{n} 1 - \sum_{i=1}^{n} 1$$
$$\iff \sum_{i=1}^{n} 1 = \sum_{i=1}^{n} 1$$
$$\iff \#\{i | b_1^i = b_2^i\} = \#\{i | b_1^i \neq b_2^i\}$$
$$\iff hdf(b_1, b_2) = n/2 \square$$
Definition 3.4: Let $B_n$ be a Hebbian base and $b_1, b \in B_n$. The $b_1$-notation of $b$ resp. $B_n$ is defined as:

$$b_1(b) := \begin{pmatrix} b_1^1 \cdot b_1^1 \\ b_1^1 \cdot b_1^2 \\ \vdots \\ b_1^1 \cdot b_n \end{pmatrix} \quad (17)$$

$$b_1(B_n) := \{b_1(b)|b \in B_n\} \quad (18)$$

Lemma 3.5: Let $b \in \mathbb{H}^n$. $B_n$ is a Hebbian base if and only if $b(B_n)$ is one also.

Proof:

$$B$$ is a Hebbian base

$$\iff \forall (b_u, b_v \in B, u \neq v) \langle b_u, b_v \rangle = 0 \quad (20)$$

$$\iff \langle b(b_u), b(b_v) \rangle = \sum_{i=1}^{n} b_i^u \cdot b_i^v = \sum_{i=1}^{n} (b_i^2 \cdot b_i^u b_i^v)$$

$$= \sum_{i=1}^{n} b_i^u b_i^v = \langle b_u, b_v \rangle = 0 \quad (21)$$

$$\iff b=b(B)$$ is a Hebbian base

Definition 3.6: If $\sigma$ is a permutation of $\{1, \ldots, n\}$, $\sigma(b), b \in \mathbb{H}^n$ is defined as:

$$\sigma(b) := \begin{pmatrix} b_{\sigma(1)} \\ b_{\sigma(2)} \\ \vdots \\ b_{\sigma(n)} \end{pmatrix} \quad (23)$$

and $\sigma(B_n)$ for a Hebbian base $B_n$:

$$\sigma(B_n) := \{\sigma(b)|b \in B_n\} \quad (24)$$

Lemma 3.7: Let $\sigma$ be a permutation as in definition 3.6. $B$ is a Hebbian base if and only if $\sigma(B)$ is one also.

Proof: Since all vectors are permuted equally the inner product is not affected.

Definition 3.8: A structure of a bipolar vector $b$ is a set $S = \{a, a+1, a+2, \ldots, e\} \subseteq \{1, 2, \ldots, n\}$ with $b^i = b^j \forall a \leq i < j \leq e$. A maximal structure of $b$ is a structure $S = \{a, a+1, a+2, \ldots, e\}$ with $b^a \neq b^{a+1}$ if $a > 1$ and $b^k \neq b^{k+1}$ if $e < n$. $S(b)$ is the set of all maximal structures of $b$. The structure size of $b$ is defined as:

$$s(b) := \min\{ord(s)|s \in S(b)\} \quad (25)$$

2) Dimension Restrictions of Hebbian Bases: At this point we are able to proof the following theorem:

Theorem 3.9: A Hebbian base $B_n$ exists only for $n = 2^k, k = 0, 1, 2, \ldots$

Proof: At first we will prove that for $n = 2^k, k = 0, 1, 2, \ldots$ a Hebbian base exists. For this purpose an algorithm is defined which generates a Hebbian base for any $n = 2^k$.

In the second step a proof for the other direction is given which uses this algorithm in a special way.

1. The algorithm in figure 2 takes any bipolar vector $b_1 \in \mathbb{H}^n, n = 2^k, k = 0, 1, 2, \ldots$ as input and returns a Hebbian base $B$ containing $b_1$. To clarify the functionality denoted in the same way, but bear in mind that only vectors with the same dimension may be compared, e.g. $a$ with $b$ but not $a$ with $c$. The progression of the algorithm is shown in figure 4. Note that not the vectors itself are drawn but their $b_1$-notation. The algorithm starts with setting $B = \{b_1\}$ (line 1). Of course $b_1$ is drawn entirely in white. In $\text{size}$ the minimal structure size over all vectors which are currently elements of $B$ is kept. As long as the size of the base $|B|$ does not reach $n$, the following steps take place.

Each vector of the base is copied to a new vector (lines 4 and 5). This new vector is modified in the following way: It is doubled. After $k$ iterations $|B| = 1$ holds. Since for every vector in $B$ a new one is added, in each iteration the size of $B$ is doubled. After $k$ iterations $|B| =
$2^k = n$ holds and the algorithm stops.

As $|B|$ doubles, size is divided by 2 at the end of each iteration. Since we assumed $n = 2^k$, at the end of the algorithm size=1 holds but in line 7 always size=0 mod 2 is valid.

So the algorithm is correct in the sense that he terminates and each step can always be performed. It remains to show that $B$ is a Hebbian base what is done by induction.

For the following considerations only the $b_1$-notations as shown in figure 4 are used but for better reading not noted. According to lemma 3.5 they are equivalent to the original vectors, i.e. if the $b_1$-notation of $B$ is a Hebbian base, $B$ itself is one also.

Let consider the state of the algorithm before the execution of line 9 and let $B' := \{b_{|B|+1}, b_{|B|+2}, \ldots, b_{2*|B|}\}$ the vectors to add. As a premise $\langle b_i, b_j \rangle = 0 \forall b_i, b_j \in B \forall b_i \neq b_j$ holds. Since this is valid for $B = \{b_1\}$ we have a starting point for the induction.

Let $b \in B, b' \in B'$. The intervals $(s, s+1, \ldots, s+\text{size} - 1)$ chosen in line 6 to 8 correspond to structures of vectors in $B$. Since this intervals were generated by splitting larger intervals from other vectors of $B$, they do not overstep structure limits for $b$, i.e. $(b^n b^{n+1} \ldots b^{n+\text{size} - 1}) = (1 1 \ldots 1)$ or $(-1 - 1 \ldots -1)$. Therefore each new generated pair of structures in $b'$ (lines 8 to 9) corresponds to a structure of $b$ leading to a hamming difference of $\text{hd}(b, b') = n/2 \Rightarrow \langle b, b' \rangle = 0$.

Furthermore $B$ is used as a pattern for generating $B'$, i.e. a structure from a vector of $b$ generates two new structures which can be imagined as a combination of two structures. There are two possible combinations: A $(+1 +1 \ldots +1)^T$ structure leads to a $((+1 +1 \ldots +1)(-1 -1 \ldots -1))^T$ combination and a $(-1 -1 \ldots -1)^T$ structure leads to a $((-1 -1 \ldots -1)(+1 +1 \ldots +1))^T$ combination. Concerning the inner product within $B'$ the resulting combinations are equivalent to the former structures. From this point of view $B'$ has the same configuration as $B$. $\Rightarrow b'_1, b'_2 \in B' \Rightarrow \langle b'_1, b'_2 \rangle = 0$. Altogether $B \cup B'$ meets the condition of the premise.

It might help to visualize the new generated pairs of structures with new colours, called white-black and black-white. A white section of $B$ is transformed into a white-black section and a black one to black-white. Since two vectors are orthogonal if their hamming difference is $\frac{n}{2}$ or $\frac{n}{2}$ by the means of their colours - if they have the same colour at the half of their elements, this transformation is leading to no change concerning the inner product.

As after $k$ iterations of the while loop $|B| = n = 2^k$ holds and together with the previous conclusion it follows that $B$ is a Hebbian base.

2. Let now an $n \neq 2^k, k = 0, 1, 2, \ldots$ be given. It will be shown that no Hebbian Base $B_n$ exists.

We start with any vector $b_1 \in H^n$ which is replaced by its $b_1$-notation as it is done for every following vector. We gain $b_1(b_1) = (1 1 \ldots 1)^T$. For every vector $b_2$, which is orthogonal to $b_1$ we can specify the set $\Sigma_0 = \{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ of all permutations which transforms $b_2$ into $\sigma_i(b_2) = (+1 +1 \ldots +1 -1 -1 \ldots -1)$ as shown in fig. 5. The applied permutation should not only affect given vectors but also ones which are added later, but for readability the permutation is not printed. At this point it is needed to decrease the structure size of a new vector $b_3$, because if not, $(-1 -1 \ldots -1 +1 +1 \ldots +1)$ would be the only possible value what is not orthogonal to $b_2$.

The inner product of $b_2$ and $b_3$ and of $b_1$ and $b_3$ has to result in 0:

$$0 = \langle b_2, b_3 \rangle = \sum_{i=0}^{n/2} b_2^i - \sum_{i=n/2+1}^{n} b_3^i$$  \hspace{1cm} \text{(26)}

$$0 = \langle b_1, b_3 \rangle = \sum_{i=0}^{n/2} b_1^i + \sum_{i=n/2+1}^{n} b_3^i$$  \hspace{1cm} \text{(27)}

Adding (26) and (27) leads to:

![Fig. 5. Possibilities for Adding New Vectors to a Hebbian Base](image)

$$0 = \sum_{i=0}^{n/2} b_3^i$$  \hspace{1cm} \text{(28)}

$$0 = \sum_{i=n/2+1}^{n} b_3^i$$  \hspace{1cm} \text{(29)}
This means that the first (second) half of \( b_3 \) is orthogonal to the first (second) half of \( b_2 \). Hence, it exists a set of permutations \( \Sigma_1 \subset \Sigma_0 \) which transforms \( b_3 \) to 
\[
\begin{align*}
+1 \ldots +1 & \quad -1 \ldots -1 \quad +1 \ldots +1 \\
n/4 \text{ times} & \quad n/4 \text{ times} & \quad n/4 \text{ times} & \quad n/4 \text{ times}
\end{align*}
\]
leaving \( b_1 \) and \( b_2 \) unchanged, because the permutations take place on a lower structure size.

At this point we can exclude \( n = 3 \), because we had to divide \( b_3 \) in 4 parts in order to add a third vector. For the vector \( b_4 \) there are two possibilities as one can see from fig. 5. With the same reason as it was used for dividing \( b_3 \) in 4 parts, \( b_5 \) has to divided in 8 parts, excluding \( n = 5, 6, 7 \) and so on.

As it was shown it is always possible to perform such permutations that the resulting vectors look like as if they were generated by the algorithm. In its non deterministic version the algorithm could generate all possible vectors (figure 6). It remains to show that the structure size has to decreased for every \( (2^k + 1) \)-th vector. We have seen that new vectors are generated as the non-deterministic algorithm would do it. Therefore the vectors \( B = \{ b_1, \ldots, b_{2^k} \} \) are a pattern for the vectors \( B' = \{ b_{2^k+1}, \ldots, b_{2^{k+1}} \} \) and if \( B \) is maximal regarding the minimal structure size, \( B' \) is it also if the maximal structures are combined to pairs as it was shown before. Since we starting with \( B = \{ b_1 \} = \{ 1 \ 1 \ 1 \ 1 \} \) which is apparently maximal regarding the minimal structure size, every \( B' \) is it. In order to get \( n \) vectors the structure size has to been halved \( \lceil \log_2 n \rceil \) times leading to a necessary dimension greater than \( n \) for \( n \neq 2^k, k = 1, 2, \ldots \).

Actually, a more general theorem was proved, namely:

**Theorem 3.10:** For any given \( n \geq 1 \), a maximum of \( 2^k \) bipolar vectors \( b_k \in \mathbb{H}^n \) is piecewise orthogonal, with \( k = \max \{ k \mid 2^k \leq n \} \).

The same proof can be used, with the difference that after the \( k \)-th decreasing of the structure size it is stopped leading to \( 2^k \) orthogonal vectors.

**D. Vector Graphics**

Since in the past only experiments with formal or natural language representations were performed, we decided to work with vector graphics. Until now we used ones with points defined by their coordinates, lines defined by their end points and circles defined by center and a point of the border.

For an example how the point \((2,4)\) is denoted as a tree take a look at figure 7. A coordinate is a node with two sons, the left one denotes the type of the father (a bullet for a coordinate) and the right one the concrete value. A point is a node with two sons also, the left one denotes again the type of the father (a cross for a point), the right one is the father of the two coordinate sub-trees. This is done analogue for the higher-level structures line and circle. Vector graphics in this form can be trained with Hebbian learning using Hebbian bases (as it was described before) resulting in \( n - 2n \) associative networks resp. matrices.

**IV. RESULTS**

Since the representation of any tree can always be given by a matrix of appropriate dimensions there is no need for experiments and we will describe only the ones with holistic computations. For more results see [7].

**A. Transformation**

Until now we worked with single points. In figure 8 one can see a grid of 25 points drawn in black and the corresponding image points shifted by \((2,1)\) and drawn in grey. All 50 points were represented by a matrix which was trained by Hebbian learning and then combined piecewise to training patterns. We used a 4 bit bipolar representation leading to \( 4 \times 8 \) matrices. The transformation network was a 32-12-32 feed-forward network which was trained with backpropagation using no activation function. With a training rate of 0.0001 and a weight decay of 0.0001 the transformation network learned all 25 patterns with a squared error over all patterns of less than 0.01 in typical 80-100 cycles. Furthermore it was able to perform the shifting operation for unknown examples with a typical squared error of 3.0 to 5.0. In spite of the relative large error the vector graphic tree could be reconstructed. For a 32-8-32 transformation network about 150 cycles were needed.
**B. Matching**

For the matching experiments we used the two series of vector graphics shown in figure 9. Each of them was trained by a $16 \times 32$ matrix. These matrices were normalized and matched by their inner product. The results are printed in table I and II. Apparently the result is nearly independent of the orientation of the graphic. This could be used for image recognition.

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.967</td>
<td>0.967</td>
<td>0.958</td>
<td>0.967</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.975</td>
<td>0.967</td>
<td>0.958</td>
<td>0.967</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.983</td>
<td>0.958</td>
<td>0.967</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.958</td>
<td>0.967</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0.992</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE I**

HEBBIAN BASES MATCHING - SERIES 1 RESULTS

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.979</td>
<td>0.979</td>
<td>0.722</td>
<td>0.698</td>
<td>0.698</td>
</tr>
<tr>
<td>2</td>
<td>0.979</td>
<td>0.698</td>
<td>0.722</td>
<td>0.698</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.698</td>
<td>0.698</td>
<td>0.722</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.972</td>
<td>0.972</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0.972</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE II**

HEBBIAN BASES MATCHING - SERIES 2 RESULTS

**V. CONCLUSION**

We have shown that structured information can be trained with Hebbian learning like it is done with RAAM networks. Furthermore the training process is much faster and - more important - always possible. The holistic transformation was possible for simple examples and it seems that holistic matching is orientation independent what is definitely worth further research.

**REFERENCES**