Draft of paper published in:
Neural Computation (2001) v. 13 n. 2, pp. 411-452

Binding and Normalization of Binary Sparse Distributed Representations by Context-Dependent Thinning

Dmitri A. Rachkovskij

V. M. Glushkov Cybernetics Center
Pr. Acad. Glushkova 40
Kiev 03680
Ukraine

dar@infrm.kiev.ua
(*Preferable contact method*)

Ernst M. Kussul

Centro de Instrumentos
Universidad Nacional Autonoma de Mexico
Apartado Postal 70186
04510 Mexico D.F.
Mexico

ekussul@servidor.unam.mx

Keywords: distributed representation, sparse coding, binary coding, binding, variable binding, representation of structure, structured representation, recursive representation, nested representation, compositional distributed representations, connectionist symbol processing.

Abstract

Distributed representations were often criticized as inappropriate for encoding of data with a complex structure. However Plate's Holographic Reduced Representations and Kanerva's Binary Spatter Codes are recent schemes that allow on-the-fly encoding of nested compositional structures by real-valued or dense binary vectors of fixed dimensionality.

In this paper we consider procedures of the Context-Dependent Thinning which were developed for representation of complex hierarchical items in the architecture of Associative-Projective Neural Networks. These procedures provide binding of items represented by sparse binary codevectors (with low probability of 1s). Such an encoding is biologically plausible and allows a high storage capacity of distributed associative memory where the codevectors may be stored.

In contrast to known binding procedures, Context-Dependent Thinning preserves the same *low* density (or sparseness) of the bound codevector for varied number of component codevectors. Besides, a bound codevector is not only similar to another one with similar component codevectors (as in other schemes), but it is also similar to the component codevectors themselves. This allows the similarity of structures to be estimated just by the overlap of their codevectors, without retrieval of the component codevectors. This also allows an easy retrieval of the component codevectors.

Examples of algorithmic and neural-network implementations of the thinning procedures are considered. We also present representation examples for various types of nested structured data (propositions using role-filler and predicate-arguments representation schemes, trees, directed acyclic graphs) using sparse codevectors of fixed dimension. Such representations may provide a fruitful alternative to the symbolic representations of traditional AI, as well as to the localist and microfeature-based connectionist representations.

1. Introduction

The problem of representing nested compositional structures is important for connectionist systems, because hierarchical structures are required for an adequate description of real-world objects and situations.

In fully local representations, an item (entity, object) of any complexity level is represented by a single unit (node, neuron) (or a set of units which has no common units with other items). Such representations are similar to symbolic ones and share their drawbacks. These drawbacks include the limitation of the number of representable items by the number of available units in the pool, and therefore, the impossibility to represent the combinatorial variety of real-world objects. Besides, a unit corresponding to a complex item only represents its name and pointers to its components (constituents). Therefore in order to determine the similarity of complex items, they should be unfolded into the base-level (indecomposable) items.

The attractiveness of distributed representations was emphasized by the paradigm of cell assemblies (Hebb, 1949) that influenced the work of Marr (1969), Willshaw (1981), Palm (1980), Hinton, McClelland & Rumelhart (1986), Kanerva (1988), and many others. In fully distributed representations, an item of any complexity level is represented by its configuration pattern over the whole pool of units. For binary units, this pattern is a subset of units which are in the active state. If the subsets corresponding to various items intersect, then the number of these subsets is much more than the number of units in the pool, providing an opportunity to solve the problem of information capacity of representations. If similar items are represented by similar subsets of units, the degree of corresponding subsets' intersection could be the measure of their similarity.

The potentially high information capacity of distributed representations provides hope for solving the problem of representing combinatorially growing number of recursive compositional items in a reasonable number of bits. Representing composite items by concatenation of activity patterns of their component items would increase the dimensionality of the coding pool. If the component items are encoded by pools of equal dimensionality, one could try to represent composite items as superposition of activity patterns of their components. The resulting coding pattern would have the same dimensionality.

However another problem arises here, known as "superposition catastrophe" (e.g. von der Malsburg, 1986) as well as "ghosts", "false" or "spurious" memory (e.g. Feldman & Ballard, 1982; Hopfield, 1982; Hopfield, Feinstein, Palmer, 1983). A simple example looks as follows. Let there be component items *a*, *b*, *c* and composite items *ab*, *ac*, *cb*. Let us represent any two of the composite items, e.g. *ac* or *cb*. For this purpose, superimpose activity patterns corresponding to the component items *a* and *c*, *c* and *b*. The ghost item *ab* also becomes represented in the result, though it is not needed. In the "superposition catastrophe" formulation, the problem consists in no way of telling which two items (*ab*, *ac*, or *ab*, *cb*, or *ac*, *cb*) make up the representation of the composite pattern *abc*, where the patterns of all three component items are activated.

The supposition of no internal structure in distributed representations (or assemblies) (Legendy, 1970; von der Malsburg, 1986; Feldman, 1989) held back their use for representation of complex data structures. The problem is to represent in the distributed fashion not only the information on the set of base-level components making up a complex hierarchical item, but also the information on the combinations in which they meet, the grouping of those combinations, etc. That is, some mechanisms were needed for binding together the distributed representations of certain items at various hierarchical levels.

One of the approaches to binding is based on temporal synchronization of constituent activation (Milner, 1974; von der Malsburg, 1981, 1985; Shastri & Ajjanagadde, 1993; Hummel & Holyoak, 1997). Though this mechanism may be useful inside single level of composition, its capabilities to represent and store complex items with multiple levels of nesting are questionable. Here we will consider binding mechanisms based on the activation of specific coding-unit subsets corresponding to a group (combination) of items, the mechanisms that are closer to the so-called conjunctive coding approach (Smolensky, 1990; Hummel & Holyoak, 1997).

"Extra units" considered by Hinton (1981) represent various combinations of active units of two or more distributed patterns. The extra units can be considered as binding units encoding various combinations of distributedly encoded items by distinct distributed patterns. Such a representation of bound items is generally described by tensor products (Smolensky, 1990) and requires an exponential growth of the number of binding units with the number of bound items. However, as was already mentioned, for recursive structures it is desired that the dimensionality of the patterns representing composite items is the same as of the component items' patterns. Besides, the most important property of distributed representations - their similarity for similar structures - should be preserved.

The problem was discussed by Hinton (1990), and a number of mechanisms for construction of his *reduced descriptions* has been proposed. Hinton (1990), Pollack (1990), Sperduti (1994) get the reduced description of a composite pattern as a result of multilevel perceptron training using back-propagation algorithm. However, their patterns are low dimensional. Plate (1991, 1995) binds high-dimensional patterns with gradual (real-valued) elements on the fly, without increase of the dimensionality, using the operation of circular convolution. Kanerva (1996) uses bitwise XOR to bind binary vectors with equal probability of 0s and 1s. Binary distributed representation are especially attractive, because binary bitwise operations are enough to handle them, providing the opportunity for significant simplification and acceleration of algorithmic implementations.

Sparse binary representations (with small fraction of 1s) are of special interest. The sparseness of codevectors allows a high storage capacity of distributed associative memories (Willshaw, Buneman, & Longuet-Higgins, 1969; Palm, 1980; Lansner & Ekeberg, 1985; Amari, 1989) which can be used for their storage, and still further acceleration of software and hardware implementations (e.g. Palm & Bonhoeffer, 1984; Kussul, Rachkovskij, & Baidyk, 1991a; Palm, 1993). Sparse encoding has also neurophysiological correlates (Foldiak & Young, 1995). The procedure for binding of sparse distributed representations ("normalization procedure") was proposed by Kussul as one of the features of the Associative-Projective Neural Networks (Kussul, 1988, 1992; Kussul, Rachkovskij, & Baidyk, 1991). In this paper, we describe various versions of such a procedure, its possible neural-network implementations, and provide examples of its use for the encoding of complex structures.

In section 2 we discuss representational problems encountered in the Associative-Projective Neural Networks and an approach for their solution. In section 3 the requirements on the Context-Dependent Thinning procedure for binding and normalization of binary sparse codes are formulated. In section 4 several versions of the thinning procedure along with their algorithmic and neural-network implementations are described. Some generalizations and notations are given in section 5. In section 6 retrieval of individual constituent codes from the composite item code is considered. In section 7 the similarity characteristics of codes obtained by Context-Dependent Thinning procedures are examined. In section 8 we show examples of encoding various structures using Context-Dependent Thinning. Related work and general discussion are presented in section 9, and conclusions are given in section 10.

2. Representation of composite items in the APNN: the problems and the answer

2.1. Features of the APNN

The Associative-Projective Neural Networks (APNN) is the name of a neural-network architecture proposed by Kussul in 1983 for the AI problems which require efficient manipulation of hierarchical data structures. Fragments of the architecture implemented in software and hardware were also used for solution of pattern-recognition tasks (Kussul, 1992, 1993). APNN features of interest here are as follows (Kussul, 1992; Kussul, Rachkovskij, & Baidyk, 1991a; Amosov et al., 1991):

- Items of any complexity (an elementary feature, a relation, a complex structure, etc.) are represented by stochastic distributed activity patterns over the neuron field (pool of units);
- The neurons are binary and therefore patterns of activity are binary vectors;
- Items of any complexity are represented over the neural fields of the same high dimensionality N;

- The number M of active neurons in the representations of items of various complexity is approximately (statistically) the same and small compared to the field dimensionality N. However M is large enough to maintain its own statistical stability.
- Items of various complexity level are stored in different distributed auto-associative neural-network memories with the same number N of neurons.

Thus, items of any complexity are encoded by sparse distributed stochastic patterns of binary neurons in the neural fields of the same dimensionality N. It is convenient to represent activity patterns in the neural fields as binary vectors, where 1s correspond to active neurons. Let us use bold-face lowercase font for codevectors to distinguish them from the items they represent denoted in italics.

The number of 1s in $\mathbf{x}$ is denoted $|\mathbf{x}|$. We seek to make $|\mathbf{x}| \approx M$ for $\mathbf{x}$ of various complexity. The similarity of codevectors is determined by the number of 1s in their intersection or overlap: $|\mathbf{x} \wedge \mathbf{y}|$, where $\wedge$ is elementwise conjunction of $\mathbf{x}$ and $\mathbf{y}$. The probability of 1s in $\mathbf{x}$ (or density of 1s in $\mathbf{x}$, or simply the vector density) is $p(\mathbf{x})=|\mathbf{x}|/N$.

Information encoding by stochastic binary vectors with a small number of 1s allows a high capacity of correlation-type neural-network memory (known as Willshaw memory or Hopfield network) to be reached using Hebbian learning rule (Wilshaw, Buneman, & Longuet-Higgins, 1969; Palm, 1980; Lansner & Ekeberg, 1985; Frolov & Muraviev, 1987, 1988; Frolov, 1989; Amari, 1989; Tsodyks, 1989). The codevectors we are talking about may be exemplified by vectors with M=100...1000, N=10 000...100 000. Though the maximal storage capacity is reached at $M=\log N$ (Willshaw, Buneman, & Longuet-Higgins, 1969; Palm, 1980), we use $M \approx \sqrt{N}$ to get a network with a moderate number N of neurons and N^2 connections at sufficient statistical stability of M (e.g. the standard deviation of M less than 3%). Under this choice of the codevector density $p=M/N \approx 1/\sqrt{N}$, the information capacity holds high enough and the number of stored items can exceed the number of neurons in the network (Rachkovskij, 1990a, 1990b; Baidyk, Kussul, & Rachkovskij, 1990).

Let us consider the problems arising in the APNN and other neural-network architectures with distributed representations when composite items are constructed.

2.2. Density of composite codevectors

The number H of component items (constituents) comprising a composite item grows exponentially with the nesting level, that is, with going to the higher levels of the part-whole hierarchy. If S items of a level l constitute an item of the adjacent higher level (l+1), then for level (l+L) the number H becomes

$$H = S^L . \quad (2.1)$$

The presence of several items comprising a composite item is encoded by the concurrent activation of their patterns, that is, by superposition of their codevectors. For binary vectors, we will use superposition by bitwise disjunction. Let us denote composite items by concatenation of symbols denoting their component items, e.g. *abc*. Corresponding composite codevectors ($a_i \vee b_i \vee c_i$, i=1,...,N) will be denoted as $\mathbf{a} \vee \mathbf{b} \vee \mathbf{c}$ or simply **abc**.

Construction of composite items will be accompanied by fast growth of density p' and respective number M' of 1s in their codevectors. For H different superimposed codevectors of low density p:

$$p'_H = 1-(1-p)^H \approx 1-e^{-pH}, \quad (2.2)$$

$$M'_H \approx p'_H N. \quad (2.3)$$

Equations 2.2 and 2.3 take into account the "absorption" of coincident 1s that prevents the exponential growth of their number versus the composition level L. However it is important that $p' >> p$ (see Figure 1) and $M' >> M$. Since the dimensionality N of codevectors representing items of various complexity is the same, the size of corresponding distributed auto-associative neural networks, where the codevectors are stored and recalled, is also the same. Therefore at $M' >> M \approx \sqrt{N}$ (at the higher levels of hierarchy) their storage capacity in terms of the number of recallable codevectors will decrease dramatically. To maintain high storage capacity at each level, M' should not substantially exceed M. However, due to the requirement of statistical stability, the number of 1s in the code can not be reduced significantly. Besides, the operation of distributed auto-associative neural-network memory usually implies the same number of 1s in codevectors. Thus it is necessary to keep the number of 1s in the codevectors of complex items approximately equal to M. (However, some variation of M between distinct hierarchical levels may be tolerable and even desirable).

These provide one of the reasons why composite items should be represented not by all 1s of their component codevectors, but only by their fraction approximately equal to M (i.e. only by some M representatives of active neurons encoding the components).

2.3. Ghost patterns and false memories

The well-known problem of ghost patterns or superposition catastrophe was mentioned in the Introduction. It consists in losing the information on the membership of component codevectors in particular composite codevector, when several composite codevectors are superimposed in their turn.

This problem is due to the essential property of superposition operation. The contribution of each member to their superposition does not depend on the contributions of other members. For superposition by elementwise disjunction, representation of **a** in $\mathbf{a}\vee\mathbf{b}$ and $\mathbf{a}\vee\mathbf{c}$ is the same. The result of superposition of several base-level component codevectors contains only the information concerning participating components and no information about the combinations in which they meet. Therefore if common items are constituents of several composite items, then the combination of the latter generally can not be inferred from their superposition codevector. For example, let a complex composite item consist of base-level items *a, b, c, d, e, f.* Then how could one determine that it really consists of the composite items *abd, bce, caf,* if there may be also other items, such as *abc, def,* etc. ? In the formulation of "false" or "spurious patterns", superposition of composite patterns **abc** and **def** generates false patterns ("ghosts") **abd, bce, caf,** etc.

The problem of introducing "false assemblies" or "spurious memories" (unforeseen attractors) into a neural network (e.g. Kussul, 1980; Hopfield, Feinstein, & Palmer, 1983; Vedenov, 1987, 1988) has the same origin as the problem of ghosts. Training of an associative memory of matrix type is usually performed using some version of Hebbian learning rule implemented by superimposing in the weight matrix the outer products of memorized codevectors. For binary connections, e.g.

$W_{ij}' = W_{ij} \vee x_i x_j$, (2.4)

where x_i and x_j are the states of the i-th and the j-th neurons when the pattern **x** to be memorized is presented (i.e. the values of the corresponding bits of **x**), W_{ij} and W_{ij}' are the connection weights between the i-th and the j-th neurons before and after training, respectively, $\vee$ stands for disjunction.

When this learning rule is sequentially used to memorize several composite codevectors with partially coinciding components, false assemblies (attractors) may appear - that is, memorized composite codevectors that were not presented to the network. For example, when representations of items *abd, bce, caf* are memorized, the false assembly *abc* (unforeseen attractor) is formed in the network (Figure 2A). Moreover, various two-item assemblies *ab*, *ad*, etc. are present, which also were not explicitly presented for storing.

The problem of introducing false assemblies can be avoided if non-distributed associative memory is used, where the patterns are not superimposed when stored and each composite codevector is placed into a separate memory word. However the problem of false patterns or superposition catastrophe still persists.

2.4. An idea of the thinning procedure

A systematic use of distributed representations provides the prerequisite to solve both the problem of codevector density growth and the superposition catastrophe. The idea of solution consists in including into the representation of a composite item not full sets of 1s encoding its component items, but only their subsets. If we choose the fraction of 1s from each component codevector so that the number of 1s in the codevector of a composite item is equal to M, then the density of 1s will be preserved in codevectors of various complexity. For example, if S=3 items of level l comprise an item of level l+1, then approximately M/3 of 1s should be preserved from each codevector of the l-th level. Then the codevector of level l+1 will have approximately M of 1s. If two items of level l+1 comprise an item of level l+2, then approximately M/2 of 1s should be preserved from each codevector of level l+1. Thus the low number M of 1s in the codevectors of composite items of various complexity is maintained, and therefore high storage capacity of the distributed auto-associative memories where these low-density codevectors are stored can be maintained as well (see also section 2.1).

Hence the component codevectors are represented in the codevector of the composite item in a reduced form - by a fraction of their 1s. The idea that the items of higher hierarchical levels ("floors") should contain their components in reduced, compressed, coarse form is well-accepted among those concerned with diverse aspects of Artificial Intelligence research. Reduced representation of component codevectors in the codevector of composite item realized in the APNN may be relevant to "coarsen models" of Amosov (1968), "reduced descriptions" of Hinton (1990), and "conceptual chunking" of Halford, Wilson, & Phillips (in press).

Reduced representation of component codevectors in the codevectors of composite items also allows a solution of the superposition catastrophe. If the subset of 1s included in the codevector of a composite item from each of the component codevectors depends on the composition of component items, then different subsets of 1s from each component codevector will be found in the codevectors of different composite items. For example, non-identical subsets of 1s will be incorporated into the codevectors of items *abc* and *acd* from **a**. Therefore the component codevectors will be bound together by the subsets of 1s delegated to the codevector of the composite item. It hinders the occurrence of false patterns and assemblies.

For the example from the Introduction, when both *ac* and *cb* are present, we will get the following overall composite codevector: $\mathbf{a_c} \vee \mathbf{c_a} \vee \mathbf{c_b} \vee \mathbf{b_c}$, where $\mathbf{x_y}$ stands for the subset of 1s in $\mathbf{x}$ that becomes incorporated into the composite codevector given $\mathbf{y}$ as the other component. Therefore if $\mathbf{a_c} \neq \mathbf{a_b}$, $\mathbf{b_c} \neq \mathbf{b_a}$, we do not observe the ghost pattern $\mathbf{a_b} \vee \mathbf{b_a}$ in the resultant codevector.

For the example of Figure 2A, where false assemblies emerge, they do not emerge under reduced representation of items (Figure 2B). Now interassembly connections are formed between different subsets of active neurons which have relatively small intersection. Therefore the connectivity of assembly corresponding to the non-presented item *abc* is low.

That the codevector of a composite item contains the subsets of 1s from the component codevectors preserves the information on the presence of component items in the composite item. That the composition of each subset of 1s depends on the presence of other component items preserves the information on the combinations in which the component items occurred. That the codevector of a composite item has approximately the same number of 1s as its component codevectors allows the combinations of such composite codevectors to be used for construction of still more complex codevectors of higher hierarchical levels.

Thus an opportunity emerges to build up the codevectors of items of varied composition level containing the information not only on the presence of their components, but on the structure of their combinations as well. It provides the possibility to estimate the similarity of complex structures without their unfolding but simply as overlap of their codevectors which is considered by many authors as a very important property for AI systems (e.g. Kussul, 1992; Hinton, 1990; Plate, 1995, 1997).

Originally the procedure reducing the sets of coding 1s of each item from the group which makes up a composite item was named "normalization" (Kussul, 1988; Kussul & Baidyk, 1990; Kussul, 1992). That name emphasized the property to maintain the number of 1s in the codes of composite items equal to that of component items.. However in this paper we will call it "Context- Dependent Thinning" (CDT) by its action mechanism, that reduces the number of 1s taking into account the context of other items from their group.

3. Requirements on the Context-Dependent Thinning procedures

Let us summarize the requirements on the CDT procedures and on the characteristics of codevectors produced by them. The procedures should process sparse binary codevectors. An important case of input is superimposed component codevectors. The procedures should output the codevector of the composite item where the component codevectors are bound and the density of the output codevector is comparable to the density of component codevectors. Let us call the resulting (output) codevector as "thinned" codevector. The requirements may be expressed as follows.

3.1. Determinism
Repeated application of the CDT procedures to the same input should produce the same output.

3.2. Variable number of inputs
The procedure should process one, two, or several codevectors. One important case of input is a vector in which several component codevectors are superimposed.

3.3. Sampling of inputs
Each component codevector of the input should be represented in the output codevector by a fraction of its 1s (or their reversible permutation).

3.4. Proportional sampling
The number of 1s representing input component codevectors in the output codevector should be proportional to their density. If the number of 1s in **a** and **b** is the same, then the number of 1s from **a** and **b** in thinned **ab** should also be (approximately) the same.

3.5. Uniform low density
The CDT procedures should maintain (approximately) uniform low density of output codevectors (small number M' of 1s) under varied number of input codevectors and their correlation degree.

3.6. Density control
The CDT procedures should be able to control the number M' of 1s in output codevectors within some range around M (the number of 1s in the component codevectors). For one important special case, $M'=M$.

3.7. Unstructured similarity
An output codevector of the CDT procedures should be similar to each component codevector at the input (or to its reversible permutation). Fulfillment of this requirement follows from fulfillment of the sampling of inputs requirement (3.3). The thinned codevector for *ab* is similar to **a** and **b.** If the densities of component codevectors are the same, the magnitude of similarity is the same (as follows from the requirement of proportional sampling, 3.4).

3.8. Similarity of subsets
The reduced representations of a given component codevector should be similar to each other to a degree that varies directly with the similarity of the set of other codevectors with which it is composed. The representation of **a** in the thinned **abc** should be more similar to its representation in the thinned **abd** than in thinned **aef**.

3.9. Structured similarity
If two sets (collections) of component items are similar, their thinned codevectors should be similar as well. It follows from the similarity of subsets requirement (3.8). If **a** and **a'** are similar, **b** and **b'** are similar, then thinned **ab** should be similar to thinned **a'b'**. Or, thinned **abc** should be similar to thinned **abd**.

3.10. Binding
Representation of a given item in an output thinned codevector should be different for different sets (collections) of component items. Representation of **a** in thinned **abc** should be different from the representation of **a** in thinned **abd**. Thus the representation of **a** in the thinned composite codevector contains information on the other components of a composite item.

4. Versions of the Context-Dependent Thinning procedures

Let us consider some versions of the CDT procedure, their properties and implementations.

4.1. Direct conjunctive thinning of two or more codevectors

Direct conjunctive thinning of binary **x** and **y** is implemented as their element-wise conjunction:

$$\mathbf{z} = \mathbf{x} \wedge \mathbf{y}, \qquad (4.1)$$

where **z** is thinned and bound result.

The requirement of determinism (section 3.1) holds for the direct conjunctive thinning procedure. The requirement of variable number of inputs (3.2) is not met, since only two codevectors are thinned. Overlapping 1s of **x** and **y** go to **z**, therefore the sampling of inputs requirement (3.3) holds. Since equal number of 1s from **x** and **y** enters into **z** even if **x** and **y** are of different density, the requirement of proportional sampling (3.4) is not fulfilled in general case.

For stochastically independent vectors **x** and **y** the density of the resulting vector **z** is:

$$p(\mathbf{z}) = p(\mathbf{x})p(\mathbf{y}) < \min(p(\mathbf{x}),p(\mathbf{y})) < 1. \qquad (4.2)$$

Here min() selects the smallest of its arguments. Let us note that for correlated **x** and **y** the density of 1s in **z** depends on the degree of their correlation. Thus $p(\mathbf{z})$ is maintained the same only for independent codevectors of constant density, and the requirement of uniform low density (3.5) is generally not met. Since $p(\mathbf{z})$ for sparse vectors is substantially lower than $p(\mathbf{x})$ and $p(\mathbf{y})$, the requirement of density control (3.6) is not met and recursive construction of bound codevectors is not supported (see also Kanerva, 1998; Sjödin, et. al., 1998). Similarity and binding requirements (3.7-3.10) may be considered as partially satisfied for two codevectors (see also Table 1).

Table 1. Properties of various versions of thinning procedures. "Yes" means that the property is present, "No" means that the property is not present, “No-Yes” and "Yes-No" mean that the property is partially present. See text for details.

Properties of thinning procedures	Direct conjunctive thinning (4.1)	Permutive thinning (4.2)	Additive (4.3) and subtractive (4.4) CDT
Determinism (3.1)	Yes	Yes	Yes
Variable number of inputs (3.2)	No-Yes	Yes	Yes
Sampling of inputs (3.3)	Yes	Yes	Yes
Proportional sampling (3.4)	No	Yes	Yes
Uniform low density (3.5)	No	No	Yes
Density control (3.6)	No	No	Yes
Unstructured similarity (3.7)	Yes-No	Yes	Yes
Similarity of subsets (3.8)	Yes-No	Yes	Yes
Structured similarity (3.9)	Yes-No	Yes	Yes
Binding (3.10)	Yes-No	Yes	Yes

Though the operation of direct conjunctive thinning of two codevectors does not meet all requirements on the CDT procedure, it has been applied by us for encoding of external information, in particular, for binding of distributed binary codevectors of feature item and its numerical value (Kussul & Baidyk, 1990; Rachkovskij & Fedoseyeva, 1990; Artykutsa et al., 1991; Kussul, Rachkovskij, & Baidyk, 1991a, 1991b). The density p of the codevectors of features and numerical values was chosen so as to provide a specified density p' of the resulting codevector (Table 2, K=2).

Table 2. The density p of K independent codevectors chosen to provide a specified density p' of codevectors produced by their conjunction.

	K										
p'	2	3	4	5	6	7	8	9	10	11	12
0.001	0.032	0.100	0.178	0.251	0.316	0.373	0.422	0.464	0.501	0.534	0.562
0.010	0.100	0.215	0.316	0.398	0.464	0.518	0.562	0.599	0.631	0.658	0.681
0.015	0.122	0.247	0.350	0.432	0.497	0.549	0.592	0.627	0.657	0.683	0.705

To thin more than two codevectors, it is natural to generalize equation 4.1:

$\mathbf{z} = \wedge_s \mathbf{x}_s$, (4.3)

where $s = 1 \ldots S$, S is the number of codevectors to be thinned. Though this operation allows binding of two or more codevectors, a single vector can not be thinned. The density of resulting codevector $\mathbf{z}$ depends on the densities of $\mathbf{x}_s$ and on their number S. Therefore to meet the requirement of uniform low density (3.5), the densities of $\mathbf{x}_s$ should be chosen depending on the number of thinned codevectors. Also, the requirement of density control (3.6) is not satisfied.

We applied this version of direct conjunctive thinning to encode positions of visual features on a two-dimensional retina. Three codevectors were bound (S=3): the codevector of a feature, the codevector of its X-coordinate, and the codevector of its Y-coordinate (unpublished work of 1991-1992 on recognition of handwritten digits, letters, and words in collaboration with WACOM Co., Japan). Also, this technique was used to encode words and word combinations for text processing (Rachkovskij, 1996). In so doing, the codevectors of letters comprising words were made bound (S>10). The density of codevectors to be bound by thinning was chosen so as to provide a specified density of the resulting codevector (Table 2, K=3...12).

Neural-network implementations of direct conjunctive thinning procedures are rather straightforward and will not be considered here.

4.2. Permutive conjunctive thinning

The codevectors to be bound by direct conjunctive thinning are not superimposed. Let us consider the case where S codevectors are superimposed by disjunction:

$\mathbf{z} = \vee_s \mathbf{x}_s$. (4.4)

Conjunction of a vector with itself produces the same vector: $\mathbf{z} \wedge \mathbf{z} = \mathbf{z}$. So let us modify $\mathbf{z}$ by permutation of all its elements and make conjunction with the initial vector:

$\mathbf{z}' = \mathbf{z} \wedge \mathbf{z}^{\sim}$. (4.5)

Here, $\mathbf{z}^{\sim}$ is the permuted vector. In vector-matrix notation, it can be rewritten as:

$\mathbf{z}' = \mathbf{z} \wedge \mathbf{P}\mathbf{z}$, (4.5a)

where $\mathbf{P}$ is an N x N permutation matrix (each row and each column of $\mathbf{P}$ has a single 1, and the rest of $\mathbf{P}$ is 0; multiplying a vector by a permutation matrix permutes the elements of the vector).

Proper permutations are those producing the permuted vector that is independent of the initial vector, e.g. random permutations or shifts. Then the density of the result is

$p(\mathbf{z}') = p(\mathbf{z})p(\mathbf{z}^{\sim}) = p(\mathbf{z})\, p(\mathbf{P}\mathbf{z})$. (4.6)

Let us consider the composition of the resulting vector:

$$\begin{aligned}
\mathbf{z}' &= \mathbf{z} \wedge \mathbf{z}^{\sim} = (\mathbf{x}_1 \vee \ldots \vee \mathbf{x}_s) \wedge \mathbf{z}^{\sim} \\
&= (\mathbf{x}_1 \vee \ldots \vee \mathbf{x}_s) \wedge (\mathbf{x}_1^{\sim} \vee \ldots \vee \mathbf{x}_s^{\sim}) \\
&= \mathbf{x}_1 \wedge (\mathbf{x}_1^{\sim} \vee \ldots \vee \mathbf{x}_s^{\sim}) \vee \ldots \vee \mathbf{x}_s \wedge (\mathbf{x}_1^{\sim} \vee \ldots \vee \mathbf{x}_s^{\sim}) \\
&= (\mathbf{x}_1 \wedge \mathbf{x}_1^{\sim}) \vee \ldots \vee (\mathbf{x}_1 \wedge \mathbf{x}_s^{\sim}) \vee \ldots \vee (\mathbf{x}_s \wedge \mathbf{x}_1^{\sim}) \vee \ldots \vee (\mathbf{x}_s \wedge \mathbf{x}_s^{\sim}). \quad (4.7)
\end{aligned}$$

Thus the resulting codevector is the superposition of all possible pairs of bitwise codevector conjunctions. Each pair includes certain component codevector and certain permuted component codevector.

Because of initial disjunction of component codevectors, this procedure meets more requirements on the CDT procedures than direct conjunctive thinning. The requirement of variable number of inputs (3.2) is now fully satisfied. As follows from equation 4.7, each component codevector $\mathbf{x}_s$ is thinned by conjunction with one and the same stochastic independent vector $\mathbf{z}^{\sim}$. Therefore statistically the same fraction of 1s is left from each component $\mathbf{x}_s$. Therefore the requirements of sampling of inputs (3.3) and of proportional sampling (3.4) hold.

For S sparse codevectors of equal density $p(\mathbf{x})<<1$

$p(\mathbf{z})\approx Sp(\mathbf{x})$, (4.8)

$p(\mathbf{z}') = p(\mathbf{z})p(\mathbf{z}^{\tilde{}}) \approx S^2p^2(\mathbf{x})$. (4.9)

To satisfy the requirements of density (3.5-3.6), $p(\mathbf{z}')=p(\mathbf{x})$ should hold for various S. It means that $p(\mathbf{x})$ should be equal to $1/S^2$. Therefore at fixed density $p(\mathbf{x})$ the density requirements (3.5-3.6) are not satisfied for variable number S of component items. The similarity and binding requirements (3.7-3.10) hold. In particular, the requirement of similarity of subsets (3.8) holds because the higher the number of identical items, the more identical conjunctions are superimposed in equation 4.7.

A neural-network implementation of permutive conjunctive thinning is shown in Figure 3. In the neural network terms, units are called "neurons", and their pools are called "neural fields". There are two input fields $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{in2}}$ and one output field $\mathbf{f}_{\text{out}}$ consisting of N binary neurons each. $\mathbf{f}_{\text{in1}}$ is connected to $\mathbf{f}_{\text{out}}$ by a bundle of N direct projective (1-to-1) connections. Each connection of this bundle connects neurons of the same number. $\mathbf{f}_{\text{in2}}$ is connected to $\mathbf{f}_{\text{out}}$ by the bundle of N permutive projective connections. Each connection of this bundle connects neurons of different numbers. "Synapses" of the neurons of the output field have weights of +1. The same pattern of superimposed component codevectors is activated in both input fields. Each neuron of the output field summarizes the excitations of its two inputs (synapses). The output is determined by comparison of the excitation level with the threshold $\theta = 1.5$. Therefore each output neuron performs conjunction of its inputs. Thus the activity pattern of the output field corresponds to bit conjunction of pattern present in the input fields and its permutation. Obviously, there are a lot of different configurations of permutive connections. Permutation by shift is particularly attractive as it is simple and fast to implement in computer simulations.

4.3. Additive CDT procedure

Though for permutive conjunctive thinning the density of resulting codevectors is closer to the density of each component codevector than for direct conjunction, it varies with the number and density of component codevectors.

Let us make a codevector $\mathbf{z}$ by the disjunction of S component codevectors $\mathbf{x}_s$, as in equation 4.4. Since the density of component codevectors is low and their number is small, then the "absorption" of common 1s is low and, according to equations 4.8 and 4.9, $p(\mathbf{z}')$ is approximately $S^2p(\mathbf{x})$ times $p(\mathbf{x})$. For example, if $p(\mathbf{x})=0.01$ and $S=5$, then $p(\mathbf{z}') \approx (1/4)p(\mathbf{x})$.

Therefore, to make the density of the thinned codevector equal to the density of its component codevectors, let us superimpose an appropriate number K of independent vectors with the density $p(\mathbf{z}')$:

$\langle\mathbf{z}\rangle = \vee_k (\mathbf{z} \wedge \mathbf{z}_k^{\tilde{}}) = \mathbf{z} \wedge (\vee_k \mathbf{z}_k^{\tilde{}})$ (4.10)

Here $\langle\mathbf{z}\rangle$ is the thinned output vector, $\mathbf{z}_k^{\tilde{}}$ is a unique (independent stochastic) permutation of elements of vector $\mathbf{z}$, fixed for each k. In vector-matrix notation, we can write:

$\langle\mathbf{z}\rangle = \vee_k (\mathbf{z} \wedge \mathbf{P}_k\mathbf{z}) = \mathbf{z} \wedge \vee_k (\mathbf{P}_k\mathbf{z})$ (4.10a)

The number K of vectors to be superimposed by disjunction can be determined as follows. If the density of the superposition of permuted versions of $\mathbf{z}$ is made

$p(\vee_k(\mathbf{P}_k\mathbf{z})) = 1/S$, (4.11)

then after conjunction with $\mathbf{z}$ (equations 4.10, 4.10a) we will get the needed density of $\langle\mathbf{z}\rangle$:

$p(\langle\mathbf{z}\rangle) = p(\mathbf{z})/S \approx Sp(\mathbf{x})/S = p(\mathbf{x})$. (4.12)

Taking into account the "absorption" of 1s in disjunction of K permuted vectors $\mathbf{z}^{\tilde{}}$, equation (4.11) can be rewritten:

$1/S = 1\text{-}(1\text{-}pS)^K$. (4.13)

Then $K=\ln(1-1/S)/\ln(1-pS)$. (4.14)

The dependence $K(S)$ at different p is shown in Table 3.

Table 3. The number K of permutations of an input codevector that produces the proper density of the thinned output codevector in the additive version of Context-Dependent Thinning. (K should be rounded to the nearest integer).

Density p of component codevector	Number S of component codevectors in the input codevector					
	2	3	4	5	6	7
0.001	346.2	135.0	71.8	44.5	30.3	21.9
0.005	69.0	26.8	14.2	8.8	6.0	4.3
0.010	34.3	13.3	7.0	4.4	2.9	2.1

4.3.1. Meeting the requirements to the CDT procedures

Since the configuration of each k-th permutation is fixed, the procedure of additive CDT is deterministic (3.1). The input vector $\mathbf{z}$ to be thinned is the superposition of component codevectors. The number of these codevectors may be variable, therefore the requirement of variable number of inputs (3.2) holds.

The output vector is obtained by conjunction of $\mathbf{z}$ (or its reversible permutation) with the independent vector $\vee_k(\mathbf{P}_k\mathbf{z})$. Therefore the 1s of all codevectors superimposed in $\mathbf{z}$ are equally represented in $\langle\mathbf{z}\rangle$ and both the sampling of inputs and the proportional sampling requirements (3.3 - 3.4) hold.

Density control of the output codevector for variable number and density of component codevectors is realized by varying K (Table 3). Therefore the density requirements (3.5 - 3.6) hold. Since the sampling of inputs and proportional sampling requirements (3.3-3.4) hold, the codevector $\langle\mathbf{z}\rangle$ is similar to all component codevectors $\mathbf{x}_s$, and the requirement of unstructured similarity (3.7) holds. The more similar are the components of one composite item to those of another, the more similar are their superimposed codevectors $\mathbf{z}$. Therefore the more similar are the vectors - disjunctions of K fixed permutations of $\mathbf{z}$, and the more similar representations of each component codevector will remain after conjunction (equation 4.10) with $\mathbf{z}$. Thus the similarity of subsets requirement (3.8) holds. Characteristics of this similarity will be considered in more detail in section 7.

Since different combinations of component codevectors produce different $\mathbf{z}$ and therefore different codevectors of K permutations of $\mathbf{z}$, representations of certain component codevector in the thinned codevector will be different for different combinations of component items, and the binding requirement (3.10) holds. The more similar are representations of each component in the output vector, the more similar are output codevectors (the requirement of structured similarity 3.9 holds).

4.3.2. An algorithmic implementation

As mentioned before, shift is an easily implementable permutation. Therefore an algorithmic implementation of this CDT procedure may be as in Figure 4A. Another example of this procedure does not require preliminary calculation of K (Figure 4B). In this version, conjunctions of the initial and permuted vectors are superimposed until the number of 1s in the output vector becomes equal to M.

4.3.3. A neural-network implementation

A neural-network implementation of the first example of the additive CDT procedure (Figure 4A) is shown in Figure 5.

To choose K depending on the density of $\mathbf{z}$, the neural-network implementation should incorporate some structures not shown in the figure. They should determine the density of the initial pattern $\mathbf{z}$ and "activate" (turn on) K bundles of permutive connections from their total number K_{max}. Alternatively, these structures should actuate the bundles of permutive connections one-by-one in the fixed order[1] until the density of the output vector in $\mathbf{f}_{out}$ becomes M/N. Let us recall that bundles of shift permutive connections are used in algorithmic implementations.

4.4. Subtractive CDT procedure

Let us consider another version of the CDT procedure. Rather than masking **z** with the straight disjunction of permuted versions of **z**, as in additive thinning, let us mask it with the inverse of that disjunction:

$\langle \mathbf{z} \rangle = \mathbf{z} \wedge \neg(\vee_k \mathbf{z}_k^{\sim}) = \mathbf{z} \wedge \neg\vee_k (\mathbf{P}_k\mathbf{z})$. (4.15)

If we choose K to make the number of 1s in $\langle \mathbf{z} \rangle$ equal to M, then this procedure will satisfy the requirements of section 3. Therefore the density of superimposed permuted versions of **z** before inversion should be $1\text{-}1/S$ (compare to equation 4.13). Thus the number K of permuted vectors to be superimposed in order to obtain the required density (taking into account "absorption" of 1s) is determined from:

$1\text{-}1/S = 1 - (1\text{-}pS)^K$. (4.16)

Then, for $pS << 1$

$K \approx \ln S/(pS)$. (4.17)

Algorithmic implementations of this subtractive CDT procedure (Kussul, 1988; Kussul & Baidyk, 1990; Kussul, 1992) are analogous to those presented for the additive CDT procedure in section 4.3.2. A neural-network implementation is shown in Figure 6.

Since the value of $\ln S/S$ is approximately the same at S=2,3,4,5 (Table 4), one can choose the K for a specified density of component codevectors $p(\mathbf{x})$ as:

$K \approx 0.34/p(\mathbf{x})$. (4.18)

At such K and S, $p(\langle \mathbf{z} \rangle) \approx p(\mathbf{x})$. Therefore the number K of permutive connection bundles in Figure 6 can be fixed, and their sequential activation is not needed. So each neuron of F_{out} may be considered as connected to an average of K randomly chosen neurons of F_{in1} by inhibitory connections. More precise values of K (obtained as exact solution of equation 4.16) for different values of p are presented in Table 4.

Table 4. The function $\ln S/S$ and the number K of permutations of an input codevector that produces the proper density of the thinned output codevector in the subtractive version of Context-Dependent Thinning procedure. (K should be rounded to the nearest integer).

	Number S of component codevectors in the input codevector					
	2	3	4	5	6	7
p			$\ln S/S$			
	0.347	0.366	0.347	0.322	0.299	0.278
0.001	346.2	365.7	345.9	321.1	297.7	277.0
0.005	69.0	72.7	68.6	63.6	58.8	54.6
0.010	34.3	36.1	34.0	31.4	29.0	26.8

This version of the CDT procedure has been originally proposed under the name "normalization procedure" (Kussul, 1988; Kussul & Baidyk, 1990; Amosov et al., 1991). We have used it in the multilevel APNN for sequence processing (Rachkovskij, 1990b; Kussul & Rachkovskij, 1991). We have also used it for binding of sparse codes in perceptron-like classifiers (Kussul, Baidyk, Lukovich, & Rachkovskij, 1993) and in one-level APNN applied for recognition of vowels (Rachkovskij & Fedoseyeva,

[1]As noted by Kanerva (personal communication), all K_{max} bundles could be activated in parallel, if the weight of the k-th bundle is set to be 2^{-k} and the common threshold of $\mathbf{f}_{int}$ neurons is adjusted dynamically so that $\mathbf{f}_{out}$ has the desired density of 1s.

1990, 1991), textures (Artykutsa et al., 1991; Kussul, Rachkovskij, & Baidyk, 1991b), shapes (Kussul & Baidyk, 1990), handprinted characters (Lavrenyuk, 1995), logical inference (Kasatkin & Kasatkina, 1991).

5. Procedures of auto-thinning, hetero-thinning, self-exclusive thinning and notation

In sections 4.2-4.4 we considered the versions of thinning procedures where a single vector (superposition of component codevectors) was the input. The corresponding pattern of activity was present both in the field $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{in2}}$ (Figures 3, 5, 6), and therefore the input vector thinned itself. Let us call these procedures "auto-thinning" or "auto-CDT" and denote them as

$^{\text{label}}\langle\mathbf{u}\rangle$. (5.1)

Here $\mathbf{u}$ is the codevector to be thinned (usually superposition of component codevectors) which is in the input fields $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{in2}}$ of Figures 3,5,6. $^{\text{label}}\langle...\rangle$ denotes particular configuration of thinning (particular realization of bundles of permutive connections). Let us note that angle brackets are used by Plate to denote normalization operation in HRRs (e.g. Plate, 1995; see also section 9.1.5).

A lot of orthogonal configurations of permutive connections are possible. Differently labeled CDT procedures implement different thinning. In the algorithmic implementations (Figure 4) different labels will use different seeds. No label corresponds to some fixed configuration of thinning. Unless otherwise specified, it is assumed that the number K of bundles is chosen to maintain the preset density of the thinned vector $\langle\mathbf{u}\rangle$, usually $|\langle\mathbf{u}\rangle|\approx M$.

$\vee_k(\mathbf{P}_k\mathbf{u})$ can be expressed as $\mathbf{Ru}$ thresholded at 1/2, where the matrix $\mathbf{R}$ is the disjunction, or it can also be the sum, of K permutation matrices $\mathbf{P}_k$. This, in turn, can be written as a function $T(\mathbf{u})$, so that we get

$\langle\mathbf{u}\rangle = \mathbf{u} \wedge T(\mathbf{u})$. (5.2)

It is possible to thin one codevector by another one if the pattern to be thinned is activated in $\mathbf{f}_{\text{in1}}$, and the pattern which thins is activated in $\mathbf{f}_{\text{in2}}$. Let us call such procedure hetero-CDT, hetero-thinning, thinning $\mathbf{u}$ with $\mathbf{w}$. We denote hetero-thinning as

$^{\text{label}}\langle\mathbf{u}\rangle_{\mathbf{w}}$. (5.3)

Here $\mathbf{w}$ is the pattern that does the thinning. It is activated in $\mathbf{f}_{\text{in2}}$ of Figures 3,5,7. $\mathbf{u}$ is the pattern which is thinned, it is activated in $\mathbf{f}_{\text{in1}}$. $^{\text{label}}\langle...\rangle$ is the configuration label of thinning. For auto-thinning, we may write $\langle\mathbf{u}\rangle = \langle\mathbf{u}\rangle_{\mathbf{u}}$.

For the additive hetero-thinning, equation 4.10 can be rewritten as

$\langle\mathbf{u}\rangle_{\mathbf{w}} = \mathbf{u} \wedge (\vee_k(\mathbf{P}_k\mathbf{w})) = \mathbf{u} \wedge T(\mathbf{w})$. (5.4)

For the subtractive hetero-thinning, equation 4.15 can be rewritten as

$\langle\mathbf{u}\rangle_{\mathbf{w}} = \mathbf{u} \wedge \neg(\vee_k(\mathbf{P}_k\mathbf{w})) = \mathbf{u} \wedge \neg T(\mathbf{w})$. (5.5)

Examples.

As before, we denote composite codevector $\mathbf{u}$ to be thinned by its component codevectors, e.g. $\mathbf{u} = \mathbf{a} \vee \mathbf{b} \vee \mathbf{c}$ or simply $\mathbf{u} = \mathbf{abc}$.

Auto-thinning of composite codevector $\mathbf{u}$:

$\langle\mathbf{u}\rangle_{\mathbf{u}} = \langle\mathbf{a}\vee\mathbf{b}\vee\mathbf{c}\rangle_{\mathbf{a}\vee\mathbf{b}\vee\mathbf{c}} = \langle\mathbf{a}\vee\mathbf{b}\vee\mathbf{c}\rangle = \langle\mathbf{abc}\rangle_{\mathbf{abc}} = \langle\mathbf{abc}\rangle$.

Hetero-thinning of composite codevector $\mathbf{u}$ with codevector $\mathbf{d}$:

$\langle\mathbf{u}\rangle_{\mathbf{d}} = \langle\mathbf{a}\vee\mathbf{b}\vee\mathbf{c}\rangle_{\mathbf{d}} = \langle\mathbf{abc}\rangle_{\mathbf{d}}$.

For both additive and subtractive CDT procedures:

$\langle\mathbf{abc}\rangle = \langle\mathbf{a}\rangle_{\mathbf{abc}} \vee \langle\mathbf{b}\rangle_{\mathbf{abc}} \vee \langle\mathbf{c}\rangle_{\mathbf{abc}}$.

We can also write $\langle\mathbf{abc}\rangle = (\mathbf{a} \wedge T(\mathbf{abc})) \vee (\mathbf{b} \wedge T(\mathbf{abc})) \vee (\mathbf{c} \wedge T(\mathbf{abc}))$. Analogous expression can be written for a composite pattern with other numbers of components. Let us note that K should be the same for thinning of the composite pattern as a whole or its individual components.

For the additive CDT procedure it is also true:

$\langle\mathbf{abc}\rangle = \langle\mathbf{a}\rangle_{\mathbf{abc}} \vee \langle\mathbf{b}\rangle_{\mathbf{abc}} \vee \langle\mathbf{c}\rangle_{\mathbf{abc}} =$

$\langle \mathbf{a}\rangle_\mathbf{a} \vee \langle \mathbf{b}\rangle_\mathbf{a} \vee \langle \mathbf{c}\rangle_\mathbf{a} \vee \langle \mathbf{a}\rangle_\mathbf{b} \vee \langle \mathbf{b}\rangle_\mathbf{b} \vee \langle \mathbf{c}\rangle_\mathbf{b} \vee \langle \mathbf{a}\rangle_\mathbf{c} \vee \langle \mathbf{b}\rangle_\mathbf{c} \vee \langle \mathbf{c}\rangle_\mathbf{c}$.

For the subtractive CDT procedure we can write:

$\langle \mathbf{a}\rangle_{\mathrm{bcd}} = \langle\langle\langle \mathbf{a}\rangle_\mathbf{b}\rangle_\mathbf{c}\rangle_\mathbf{d}$ and

$\langle \mathbf{abc}\rangle = \langle\langle\langle \mathbf{a}\rangle_\mathbf{a}\rangle_\mathbf{b}\rangle_\mathbf{c} \vee \langle\langle\langle \mathbf{b}\rangle_\mathbf{a}\rangle_\mathbf{b}\rangle_\mathbf{c} \vee \langle\langle\langle \mathbf{c}\rangle_\mathbf{a}\rangle_\mathbf{b}\rangle_\mathbf{c}$.

Let us also consider a modification of the auto-CDT procedures which will be used in section 7.2. If we eliminate the thinning of a component codevector with itself, we obtain "self-exclusive" auto-thinning. Let us denote it as $\langle \mathbf{abc}\rangle_{\backslash\mathbf{abc}}$:

$\langle \mathbf{abc}\rangle_{\backslash\mathbf{abc}} = \langle \mathbf{a}\rangle_\mathbf{bc} \vee \langle \mathbf{b}\rangle_\mathbf{ac} \vee \langle \mathbf{c}\rangle_\mathbf{ab}$.

6. Retrieval of component codevectors

After thinning, the codevectors of component items are present in the thinned codevector of a composite item in a reduced form. We must be able to retrieve complete component codevectors. Since the requirement of the unstructured similarity (3.7) holds, the thinned composite codevector is similar to its component codevectors. So if we have a full set (alphabet) of component codevectors of the preceding (lower) level of compositional hierarchy, we can compare them with the thinned codevector. The similarity degree is determined by the overlap of codevectors. The alphabet items corresponding to the codevectors with maximum overlaps are the sought-for components.

The search of the most similar component codevectors can be performed by a sequential finding of overlaps of the codevector to be decoded with all codevectors of the component alphabet. An associative memory can be used to implement this operation in parallel. After retrieving of the full-sized component codevectors of the lower hierarchical level, one can then retrieve their component codevectors of still lower hierarchical level in an analogous way. For this purpose, the alphabet of the latter should be known as well. If the order of component retrieval is important, some auxiliary procedures can be used (Kussul, 1988; Amosov et al., 1991; Rachkovskij, 1990b; Kussul & Rachkovskij, 1991).

Example. Let us consider the alphabet of six component items a, b, c, d, e, f. They are encoded by stochastic fixed vectors of N=100000 bits with M≈1000 bits set to 1. Let us obtain the thinned codevector $\langle \mathbf{abc}\rangle$. The number of 1s in $\langle \mathbf{abc}\rangle$ in our numerical example is $|\langle \mathbf{abc}\rangle| = 1002$. Let us find the overlap of each component codevector with the thinned codevector: $|\mathbf{a} \wedge \langle \mathbf{abc}\rangle| = 341$; $|\mathbf{b} \wedge \langle \mathbf{abc}\rangle| = 350$; $|\mathbf{c} \wedge \langle \mathbf{abc}\rangle| = 334$; $|\mathbf{d} \wedge \langle \mathbf{abc}\rangle| = 12$; $|\mathbf{e} \wedge \langle \mathbf{abc}\rangle| = 7$; $|\mathbf{f} \wedge \langle \mathbf{abc}\rangle| = 16$. So the representation of the component items a, b, c is substantially higher than the representation of the items d, e, f occurring due to a stochastic overlap of independent binary codevectors. The numbers obtained are typical for the additive and the subtractive versions of thinning, as well as for their self-exclusive versions.

7. Similarity preservation by the thinning procedures

In this section, let us consider the similarity of thinned composite codevectors as well as the similarity of thinned representations of component codevectors in the thinned composite codevectors. These kinds of similarity are considered under different combinations of component items and different versions of thinning procedures.

Let us use the following Context-Dependent Thinning procedures:

- permutive conjunctive thinning, section 4.2 (Paired-M);
- additive auto-CDT, section 4.3 (CDTadd);
- additive self-exclusive auto-CDT, section 5 (CDTadd-sl);
- subtractive auto-CDT, section 4.4 (CDTsub);
- subtractive self-exclusive auto-CDT, section 5 (CDTsub-sl).

For these experiments, let us first obtain the composite codevectors which have 5 down to 0 component codevectors in common: **abcde**, **abcdf**, **abcfG**, **abfgh**, **afghi**, **fghij.** For the component codevectors, N=100000 bits with M≈1000. Then, for each thinning procedure, let us thin the composite codevectors down to the density of their component codevectors. (For permutive conjunctive thinning, the density of component codevectors was chosen to get approximately M of 1s in the result).

7.1. Similarity of thinned codevectors

Let us find an overlap of thinned codevector $\langle \mathbf{abcde} \rangle$ with $\langle \mathbf{abcde} \rangle$, $\langle \mathbf{abcdf} \rangle$, $\langle \mathbf{abcfg} \rangle$, $\langle \mathbf{abfgh} \rangle$, $\langle \mathbf{afghi} \rangle$, and $\langle \mathbf{fghij} \rangle$. Here $\langle\rangle$ is used to denote any thinning procedure. A normalized measure of the overlap of **x** with various **y** is determined as $|\mathbf{x} \wedge \mathbf{y}|/|\mathbf{x}|$.

The experimental results are presented in Figure 7A, where the normalized overlap of thinned composite codevectors is shown versus the normalized overlap of corresponding unthinned composite codevectors. It can be seen that the overlap of thinned codes for various versions of the CDT procedure is approximately equal to the square of overlap of unthinned codes. For example, the similarity (overlap) of **abcde** and **abfgh** is approximately 0.4 (two common components of five total), and the overlap of their thinned codevectors is about 0.16.

7.2. Similarity of component codevector subsets included into thinned codevectors

Some experiments were conducted in order to investigate the similarity of subsets requirement (3.8). The similarity of subsets of a component codevector incorporated into various thinned composite vectors was obtained as follows. First, the intersections of various thinned five-component composite codevectors with their component **a** were determined: $\mathbf{u}=\mathbf{a} \wedge \langle \mathbf{abcde} \rangle$, $\mathbf{v}=\mathbf{a} \wedge \langle \mathbf{abcdf} \rangle$, $\mathbf{w}=\mathbf{a} \wedge \langle \mathbf{abcfg} \rangle$, $\mathbf{x}=\mathbf{a} \wedge \langle \mathbf{abfgh} \rangle$, $\mathbf{y}=\mathbf{a} \wedge \langle \mathbf{afghi} \rangle$. Then, the normalized values of the overlap of intersections were obtained as $|\mathbf{u} \wedge \mathbf{v}|/|\mathbf{u}|$, $|\mathbf{u} \wedge \mathbf{w}|/|\mathbf{u}|$, $|\mathbf{u} \wedge \mathbf{x}|/|\mathbf{u}|$, $|\mathbf{u} \wedge \mathbf{y}|/|\mathbf{u}|$.

Figure 7B shows how the similarity (overlap) of component codevector subsets incorporated into two thinned composite codevectors varies versus the similarity of corresponding unthinned composite codevectors. It can be seen that these dependencies are different for different thinning procedures: for the CDTadd and the CDTadd-sl they are close to linear, but for the CDTsub and CDTsub-sl they are polynomial. Which is preferable, depends on the application.

7.3. The influence of the depth of thinning

By the depth of thinning we understand the density value of a thinned composite codevector. Before, we considered it equal to the density of component codevectors. Here, we vary the density of the thinned codevectors. The experimental results presented in Figure 8 are useful for the estimation of resulting similarity of thinned codevectors in applications.

As in sections 7.1-7.2, composite codevectors of five components were used. Therefore approximately $5M$ of 1s (actually, more close to $4.9M$ because of random overlaps) were in the input vector before thinning. We varied the number of 1s in the thinned codevectors from $4M$ to $M/4$. Only the additive and the subtractive CDT procedures were investigated.

The similarity of thinned codevectors is shown in Figure 8A. For a shallow thinning, where the resulting density is near the density of input composite codevector, the similarity degree of resulting vectors is close to that of input codevectors (the curve is close to linear). For a deep thinning, where the density of thinned codevectors is much less than the density of input codevectors, the similarity function behave as a power function, transforming from linear through quadratic to cubic (for subtractive thinning).

The similarity of component subsets in the thinned codevector is shown in Figure 8B. For the additive CDT procedure, the similarity function is linear, and its angle reaches approximately 45° for "deep" thinning. For the subtractive CDT procedure, the function is similar to the additive one for the "shallow" thinning, and becomes near-quadratic for the "deep" thinning.

8. Representation of structured expressions

Let us consider representation of various kinds of structured data by binary sparse codevectors of fixed dimensionality. In the examples below, the items of the base level for a given expression. In its turn, they may, in their turn, represent complex structured data.

8.1. Transformation of symbolic bracketed expressions into representations by codevectors

Performing the CDT procedure can be viewed as analog of introducing brackets into symbolic descriptions. As mentioned in section 5, the CDT procedures with different thinning configurations are denoted by different labels at the opening thinning bracket: $^1\langle\ \ \rangle$, $^2\langle\ \ \rangle$, $^3\langle\ \ \rangle$, $^4\langle\ \ \rangle$, $^5\langle\ \ \rangle$, etc.

Therefore in order to represent a complex symbolic structure by a distributed binary codevector, one should

- map each symbol of the base-level item to the corresponding binary sparse codevector of fixed dimensionality;
- replace conventional brackets in symbolic bracketed representation by "thinning" ones. Each compositional level has its own label of thinning brackets, that is, thinning configuration;
- superimpose the codevectors inside the thinning brackets of the deepest nesting level by elementwise disjunction;
- perform the CDT procedure on superimposed codevectors using the configuration of thinning corresponding to particular "thinning" label;
- superimpose the resulting thinned vectors inside the thinning brackets of the next nesting level;
- perform the CDT procedure on superimposed codevectors using appropriate thinning configuration of that nesting level;
- repeat the two previous steps until the whole structure is encoded.

8.2. Representation of ordered items

For many propositions, the order of arguments is essential. To represent the order of items encoded by the codevectors, binding with appropriate roles is usually used.

One approach is to use explicit binding of role codevectors (agent-object, antecedent-consequent, or just ordinal number) with the item (filler) codevector. This binding can be realized by an auto- or hetero-CDT procedure (Rachkovskij, 1990b). The item a which is #3 may be represented as $\langle \mathbf{a} \vee \mathbf{n3} \rangle$ or $\langle \mathbf{a} \rangle_{\mathbf{n3}}$, where **n3** is the codevector of the “third place” role.

Another approach is to use implicit binding by providing different locations for different positions of an item in a proposition. To preserve fixed dimensionality of codevectors, it was proposed to encode different positions by the specific shifts of codevectors (Kussul & Baidyk, 1993). (Reversible permutations can be also used). For our example, we have **a** shifted by the number $n3$ of 1-bit shifts corresponding to the “third place” of an item.

These and other techniques to represent the order of items have their pros and cons. Thus a specific technique should be chosen depending on the application. So we will not consider details here. It is important that such techniques exist, and we will denote the codevector of item a at the n-th place simply by $\mathbf{a}\_n$.

Let us note that generally the modification of an item codevector to encode its ordinal number should be different for different nesting levels. It is analogous to having its own thinning configuration at each level of nesting. Therefore **a** and **b** should be modified in the same manner in $^1\langle ...\mathbf{a}\_n... \rangle$ and $^1\langle ...\mathbf{b}\_n... \rangle$, but **a** should generally be modified differently in $^2\langle ...\mathbf{a}\_n... \rangle$.

8.3. Examples

8.3.1. Role-filler structure

Representations of structures or propositions by the role-filler scheme are widely used (Smolensky, 1990; Pollack, 1990; Plate, 1991, 1995; Kanerva, 1996; Sperduti, 1994). Let us consider the relational instance

knows(Sam, loves(John, Mary)). (8.1)
Using the Holographic Reduced Representations of Plate, it can be represented as:
L1 = love + loveagt∗john + loveobj∗mary, (8.2)
L2 = know + knowagt∗sam + knowobj∗L1, (8.3)
where ∗ stands for binding operation, and + denotes addition. In our representation:
$\mathbf{L1} = {}^{2}\langle \mathbf{love} \vee {}^{1}\langle \mathbf{loveagt} \vee \mathbf{john} \rangle \vee {}^{1}\langle \mathbf{loveobj} \vee \mathbf{mary} \rangle\rangle$, (8.4)
$\mathbf{L2} = {}^{4}\langle \mathbf{know} \vee {}^{3}\langle \mathbf{knowagt} \vee \mathbf{sam} \rangle \vee {}^{3}\langle \mathbf{knowobj} \vee \mathbf{L1} \rangle\rangle$. (8.5)

8.3.2. Predicate-arguments structure

Let us consider representation of relational instances *loves(John, Mary)* and *loves(Tom, Wendy)* by the predicate-arguments (or symbol-argument-argument) structure (Halford, Wilson, & Phillips, in press):
loves∗John∗Mary + loves∗Tom∗Wendy. (8.6)
Using our representation, we obtain:
${}^{2}\langle {}^{1}\langle \mathbf{loves}\_0 \vee \mathbf{John}\_1 \vee \mathbf{Mary}\_2 \rangle \vee {}^{1}\langle \mathbf{loves}\_0 \vee \mathbf{Tom}\_1 \vee \mathbf{Wendy}\_2 \rangle\rangle$. (8.7)
Let us note that this example may be represented using the role-filler scheme of HRRs as
L1 = loves + lover∗Tom + loved∗Wendy, (8.8)
L2 = loves + lover∗John + loved∗Mary, (8.9)
L = L1 + L2. (8.10)
Under such a representation, the information about who loves whom is lost in **L** (Plate, 1995; Halford, Wilson, & Phillips, in press). In our representation, this information is preserved even using the role-filler scheme:
$\mathbf{L1} = {}^{2}\langle \mathbf{loves} \vee {}^{1}\langle \mathbf{lover} \vee \mathbf{Tom} \rangle \vee {}^{1}\langle \mathbf{loved} \vee \mathbf{Wendy} \rangle\rangle$, (8.11)
$\mathbf{L2} = {}^{2}\langle \mathbf{loves} \vee {}^{1}\langle \mathbf{lover} \vee \mathbf{John} \rangle \vee {}^{1}\langle \mathbf{loved} \vee \mathbf{Mary} \rangle\rangle$, (8.12)
$\mathbf{L} = \langle \mathbf{L1} \vee \mathbf{L2} \rangle$. (8.13)
Another example of relational instance from Halford, Wilson, & Phillips (in press):
cause(shout-at(John ,Tom),hit(Tom, John)). (8.14)
Using our representation scheme, it may be represented as
${}^{2}\langle \mathbf{cause}\_0 \vee {}^{1}\langle \mathbf{shout\text{-}at}\_0 \vee \mathbf{John}\_1 \vee \mathbf{Tom}\_2 \rangle\_1 \vee {}^{1}\langle \mathbf{hit}\_0 \vee \mathbf{Tom}\_1 \vee \mathbf{John}\_2 \rangle\_2 \rangle$. (8.15)

8.3.3. Tree-like structure

An example of bracketed binary tree adapted from Pollack (1990):
((d (a n)) (v (p (d n)))). (8.16)
If we do not take the order into account, but use only the information about the grouping of constituents, our representation may look as simple as:
${}^{4}\langle {}^{3}\langle \mathbf{d} \vee {}^{2}\langle \mathbf{a} \vee \mathbf{n} \rangle\rangle \vee {}^{3}\langle \mathbf{v} \vee {}^{2}\langle \mathbf{p} \vee {}^{1}\langle \mathbf{d} \vee \mathbf{n} \rangle\rangle\rangle\rangle$. (8.17)

8.3.4 Labeled directed acyclic graph

Sperduti & Starita (1997), Frasconi, Gori, & Sperduti (1997) provide examples of labeled directed acyclic graphs. Let us consider
F(a, f(y), f(y, F(a, b))). (8.18)
Using our representation, it may look as
${}^{3}\langle \mathbf{F}\_0 \vee \mathbf{a}\_1 \vee {}^{2}\langle \mathbf{f}\_0 \vee \mathbf{y}\_1 \rangle\_2 \vee {}^{2}\langle \mathbf{f}\_0 \vee \mathbf{y}\_1 \vee {}^{1}\langle \mathbf{F}\_0 \vee \mathbf{a}\_1 \vee \mathbf{b}\_2 \rangle\_2 \rangle\_3 \rangle$. (8.19)

9. Related work and discussion

The procedures of Context-Dependent Thinning allow construction of binary sparse representations of complex data structures, including nested compositional structures or part-whole hierarchies. The basic principles of such representations and their use for data handling were proposed in the context of the

Associative-Projective Neural Networks paradigm (Kussul, 1988, 1992; Kussul, Rachkovskij, & Baidyk, 1991a).

9.1 Comparison to other representation schemes

Let us compare our scheme for representation of complex data structures using the CDT procedure (we will call it "APNN-CDT" below) with other schemes using distributed representations. The best known schemes are (L)RAAMs (Pollack, 1990; Blair, 1997; Sperduti 1994), Tensor Product representations (Smolensky, 1990; Halford, Wilson, & Phillips, in press), Holographic Reduced Representations (HRRs) (Plate, 1991, 1995), Binary Spatter Codes (BSCs) (Kanerva, 1994, 1996). For this comparison, we will use the framework of Plate (1997) who proposes to distinguish these schemes using the following features: the nature of distributed representation; the choice of superposition; the choice of binding operation; how the binding operation is used to represent predicate structure; the use of other operations and techniques.

9.1.1. The nature of distributed representation

Vectors of random real-valued elements with the Gaussian distribution are used in HRRs. Dense binary random codes with the number of 1s equal to the number of 0s are used in BSCs. Vectors with real or binary elements (without specified distributions) are used in other schemes.

In the APNN-CDT scheme, binary vectors with randomly distributed small number of 1s are used to encode base-level items.

9.1.2. The choice of superposition

The operation of superposition is used for unstructured representation of an aggregate of codevectors.

In BSCs superposition is realized as a bitwise thresholded addition of codevectors. Schemes with non-binary elements, such as HRRs, use elementwise summation. For tensors, superposition is realized as adding up or ORing the corresponding elements.

In the APNN-CDT scheme, elementwise OR is used.

9.1.3. The choice of binding operation

Most schemes use special operations for binding of codevectors. The binding operations producing the bound vector that has the same dimension as initial codevectors (or one of them in (L)RAAMs) are convenient for representation of recursive structures. The binding operation is performed "on the fly" by circular convolution (HRRs), elementwise multiplication (Gayler, 1998), or XOR (BSCs). In (L)RAAMs, binding is realized through multiplication of input codevectors by the weight matrix of the hidden layer formed by training of a multilayer perceptron using the codevectors to be bound.

The vector obtained by binding can be bound with another codevector in its turn. In Tensor Models, binding of several codevectors is performed by their tensor product. The dimensionality of resulting tensor grows with the number of bound codevectors.

In the APNN-CDT scheme, binding is performed by the Context-Dependent Thinning procedure. Unlike the other schemes where the codevectors to be bound are not superimposed, they can be superimposed by disjunction in the basic version of the CDT procedure. Superposition codevector **z** (as in equation 4.4) makes the context codevector. The result of the CDT procedure may be considered as superimposed bindings of each component codevector with the context codevector. Or, it may be considered as superimposed paired bindings of all component codevectors with each other. (Note that in the “self-exclusive” CDT version (section 5) the codevector of each component is not bound to itself. In the hetero-CDT version, one codevector is bound to another codevector through thinning with the latter).

According to Plate's framework, CDT as a binding procedure can be considered as a special kind of superposition (disjunction) of certain elements of the tensor product of **z** by itself (i.e. N^2 scalar products $z_i z_j$). Actually, $\langle z_i \rangle$ is disjunction of certain $z_i z_j = z_i \wedge z_j$, where z_j is the j-th element of permuted **z** (equation

4.10). CDT can be also considered as a hashing procedure: the subspace to where hashing is performed is defined by 1s of **z**, and some 1s of **z** are mapped to that subspace.

Since the resulting bound codevector $\langle \mathbf{z} \rangle$ is obtained in the CDT procedure by thinning the 1s of **z,** (where the component codevectors are superimposed), $\langle \mathbf{z} \rangle$ is similar to its component codevectors (unstructured similarity is preserved). Therefore to retrieve the components bound in the thinned codevector, we only need to choose the most similar component codevectors from their alphabet. This can be done using an associative memory.

None of the mentioned binding operations, except for the CDT, preserves unstructured similarity. Therefore to extract some component codevector from the bound codevector, they demand to know the other component codevector(s). Then rebinding of the bound codevector with the inverses of known component codevector(s) produces a noisy version of the sought component codevector. This operation is known as decoding or "unbinding". To eliminate noise from the unbound codevector, a "clean-up" memory with the full alphabet of component codevectors is also required in those schemes. If some or all components of the bound codevector are not known, decoding in those schemes requires exhaustive search (substitution, binding, and checking) through all combinations of codevectors from the alphabet. Then the obtained bound codevector most similar to the bound codevector to be decoded provides the information on its composition.

As in the other schemes, structured similarity is preserved by the CDT, i.e. bindings of similar patterns are similar to each other. However the character of similarity is different. In most of the other schemes the similarity of the bound codevectors is equal to the product of similarities of the component codevectors (e.g. Plate, 1995). For example, the similarity of $\mathbf{a}*\mathbf{b}$ and $\mathbf{a}*\mathbf{b'}$ is equal to the similarity of **b** and **b'**. Therefore if **b** and **b'** are not similar at all, the bound vectors also will not be similar.

The codevectors to be bound by the CDT procedure are initially superimposed component codevectors. So their initial similarity is the mean of the components' similarities. Also, the thinning itself preserves approximately the square of similarity of the input vectors. So, the similarity for dissimilar **b** and **b'** will be >0.25 instead of 0 for the other schemes.

9.1.4. How the binding operation is used to represent predicate structure

In most of the schemes predicate structures are represented by role-filler bindings. Halford, Wilson, & Phillips (in press) use predicate-argument bindings. The APNN-CDT scheme allows such representations of predicate structures as role-filler bindings, predicate-argument bindings, and also offers a potential for other possible representations. Both ordered and unordered arguments can be represented.

9.1.5. The use of other operations and techniques.

- *Normalization.* After superposition of codevectors, some normalizing transformation is used in various schemes to bring the individual elements or the total strength of the resulting codevector within certain limits. In BSCs, it is the threshold operation that converts a non-binary codevector (the bitwise sum of component codevectors) to a binary one. In HRRs, it is the scaling of codevectors to the unit length that facilitates their comparison.

The CDT procedure performs a dual role: it not only binds superimposed codevectors of components, but it also normalizes the density of the resulting codevector. It would be interesting to check to what extent the normalization operations in other schemes provide the effect of binding as well.

Clean-up memory. Associative memory is used in various representation schemes for storage of component codevectors and their recall (clean-up after finding their approximate noisy versions using unbinding). After the CDT procedure, the resulting codevector is similar to its component codevectors, however the latter are represented in the reduced form. Therefore it is natural to use associative memories in the APNN-CDT scheme to store and retrieve the codevectors of component items of various complexity levels. Since component codevectors of different complexity levels have approximately the same and small number of 1s, an associative memory based on assembly neural networks with simple Hebbian learning rule allows efficient storage and retrieval of a large number of codevectors.

Chunking. The problem of chunking remains one of the least developed issues in existing representation schemes.

In the HRRs and BSCs chunks are normalized superpositions of stand-alone component codevectors and their bindings. In its turn, the codevector of a chunk can be used as one of the components for binding. Thus, chunking allows structures of arbitrary nesting or composition level to be built. Each chunk should be stored in a clean-up memory. When complex structures are decoded by unbinding, noisy versions of chunk codevectors are obtained. They are used to retrieve pure versions from the clean-up memory, which can be decoded in their turn.

In those schemes, the codevectors of chunks are not bound. Therefore they can not be superimposed without the risk of structure loss, as it was repeatedly mentioned in this paper. In the APNN-CDT scheme, any composite codevector after thinning represents a chunk. Since the component codevectors are bound in the chunk codevector, the latter can be operated as a single whole (an entity) without confusion of components belonging to different items.

When a compositional structure is constructed using HRRs or BSCs, the chunk codevector is usually the filler which becomes bound with some role codevector. In this case, in distinction to the APNN-CDT scheme, the components **a**, **b**, **c** of the chunk become bound with the role rather than with each other:

$\mathbf{role}*(\mathbf{a} + \mathbf{b} + \mathbf{c}) = \mathbf{role}*\mathbf{a} + \mathbf{role}*\mathbf{b} + \mathbf{role}*\mathbf{c}$. (9.1)

Again, if the role is not unique, it can not be determined to which chunk the binding $\mathbf{role}*\mathbf{a}$ belongs. Also, the role codevector should be known for unbinding and subsequent retrieving of the chunk.

Thus in the representation schemes of HRRs and Binary Spatter Codes each of the component codevectors belonging to a chunk binds with (role) codevectors of other hierarchical levels not belonging to that chunk. Therefore such bindings may be considered as "vertical". In the APNN-CDT scheme, a "horizontal" binding is essential: the codevectors of the chunk components are bound with each other.

In the schemes of Plate, Kanerva, and Gayler, the vertical binding chain **role_upper_level** $*$ (**role_lower_level** $*$ **filler**) is indistinguishable from **role_lower_level** $*$ (**role_upper_level** $*$ **filler**),because their binding operations are associative and commutative. For the CDT procedure, in contrast, ${}^2\langle{}^1\langle\mathbf{a} \vee \mathbf{b}\rangle \vee \mathbf{c}\rangle \neq {}^2\langle\mathbf{a} \vee {}^1\langle\mathbf{b} \vee \mathbf{c}\rangle\rangle$, and also $\langle\langle\mathbf{a} \vee \mathbf{b}\rangle \vee \mathbf{c}\rangle \neq \langle\mathbf{a} \vee \langle\mathbf{b} \vee \mathbf{c}\rangle\rangle$.

Gayler (1998) proposes to bind a chunk codevector with its permuted version. It resembles the version of thinning procedure from section 4.2, but for real-valued codevectors. Different codevector permutations for different nesting levels allow the components of chunks from different levels to be distinguished, in a similar fashion as using different configurations of thinning connections in the CDT. However since the result of binding in the scheme of Gayler and in the other considered schemes (with the exception of APNN-CDT) is not similar to the component codevectors, in those schemes decoding of the chunk codevector created by binding with a permutation of itself will generally require exhaustion of all combinations of component codevectors.

This problem with the vertical binding schemes of Plate, Kanerva, and Gayler can be rectified by using a binding operation that, prior to a conventional binding operation, permutes its left and right arguments differently (as discussed on p. 84 in Plate (1994)).

The obvious problem of Tensor Product representation is the growth of dimensionality of the resulting pattern obtained by the binding of components. If it is not solved, the dimensionality will grow exponentially with the nesting depth. Halford, Wilson, & Phillips (in press) consider chunking as the means to reduce the rank of tensor representation. To realize chunking, they propose to use the operations of convolution, concatenation, superposition, as well as some special function that associates the outer product with the codevector of lower dimension. However the first three operations do not rule out confusion of grouping or ordering of arguments inside chunk, (i.e., different composite items may produce identical chunks). And the special function (and its inverse) requires concrete definition. Probably it could be done using associative memory, e.g. of the sigma-pi type proposed by Plate (1998).

In (L)RAAMs the chunks of different nesting levels are encoded in the same weight matrix of connections between the input layer and the hidden layer of a multilayer perceptron. It may be one of the reasons for poor generalization. Probably if additional multilayer perceptrons are introduced for each nesting level (with the input for each following perceptron provided by the hidden layer of the preceding one, similarly to Sperduti & Starita, 1997), generalization in those schemes would improve.

In the APNN, chunks (thinned composite codevectors) of different nesting levels are memorized in different auto-associative neural networks. It allows an easy similarity-based decoding of a chunk through its subchunks of the previous nesting level and decreases memory load at each nesting level (see also Rachkovskij, accepted).

9.2. Sparse binary schemes

Indicating unknown areas where useful representation schemes for nested compositional structures can be found, Plate (1997) notes that known schemes poorly handle sparse binary patterns, because known binding and superposition operations change the density of sparse patterns.

From the work known to us, only Sjödin (1998) expresses the idea of "thinning" in an effort to avoid the low associative memory capacity for dense binary patterns. He defines the thinning operation as preserving the 1s corresponding to the maximums of some function defined over the binary vector. The values of that function can be determined at cyclic shifts of the codevector by the number of steps equal to the ordering number of 1s in that codevector. However it is not clear from such a description what are the properties of the maximums and, therefore, what is the character of similarity.

The CDT procedure considered in this paper allows the density of codevectors to be preserved while binding them. Coupled with the techniques for encoding of the pattern ordering, this procedure allows implementation of various representation schemes of complex structured data. Approximately the same low density of binary codevectors at different nesting levels permits the use of identical procedures for construction, recognition, comparison, and decoding of patterns at different hierarchical levels of the Associative-Projective Neural Network architecture (Kussul 1988, 1992; Kussul, Rachkovskij, & Baidyk, 1991).

The CDT procedure preserves the similarity of encoded descriptions allowing the similarity of complex structures to be determined by the overlap of their codevectors. Also, in the codevectors of complex structures formed using the CDT procedure, representation of the component codevectors (subset of their 1s) is reduced. Therefore, the APNN-CDT scheme can be considered another implementation of Hinton's (1990) reduced descriptions, Amosov's (1967) item coarsing, or compression of Halford, Wilson, & Phillips (in press). Besides, the CDT scheme is biologically relevant since it uses sparse representations and allows simple neural-network implementation.

10. Conclusion

Procedures of Context-Dependent Thinning described in the paper perform binding of items represented as sparse binary codevectors. They allow a variable number of superimposed patterns to be bound on the fly while preserving the density of bound codevectors. The result of the CDT is of the same dimensionality as the component codevectors. Using of the auto-CDT procedures as analogs of brackets in the bracketed symbolic representation of various complex data structures permits easy transformation of these representations to the binary codevectors of fixed dimensionality with a small number of 1s.

Unlike other binding procedures, binding by the auto-CDT preserves the similarity of bound codevector with each of the component codevectors. It makes possible both to determine the similarity of complex items with each other by the overlap of their codevectors and to retrieve in full size the codevectors of their components. Such operations are efficiently implementable by distributed associative memories which provide high storage capacity for the codevectors with small number of 1s.

The APNN-CDT style representations have been already used by us in applications (an earlier work is reviewed in Kussul, 1992, 1993; more recent developments are described in Lavrenyuk, 1995; Rachkovskij, 1996; Kussul & Kasatkina, 1999; Rachkovskij, accepted). We hope that the CDT procedures will find their application for distributed representation and manipulation of complex compositional data structures, contributing to the progress of connectionist symbol processing (Touretzky, 1995, 1990; Touretzky & Hinton, 1988; Hinton, 1990; Plate, 1995). Fast (parallel) evaluation of similarity or finding the most similar compositional items allowed by such representations are extremely useful for solution of a wide range of AI problems.

Acknowledgements: The authors are grateful to Pentti Kanerva and Tony Plate for their extensive and helpful comments, valuable suggestions, and continuous support. This work was funded in part by the International Science Foundation Grants U4M000 and U4M200.

References

Amari, S. (1989). Characteristics of sparsely encoded associative memory. *Neural Networks, 2*, 445-457.
Amosov, N. M. (1967). *Modelling of thinking and the mind*. New York: Spartan Books.
Amosov, N. M., Baidyk, T. N., Goltsev, A. D., Kasatkin, A. M., Kasatkina, L. M., Kussul, E. M., & Rachkovskij, D. A. (1991). *Neurocomputers and intelligent robots*. Kiev: Naukova dumka. (In Russian).
Artykutsa, S. Ya., Baidyk, T. N., Kussul, E. M., & Rachkovskij, D. A. (1991). Texture recognition using neurocomputer. (Preprint 91-8). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).
Baidyk, T. N., Kussul, E. M., & Rachkovskij, D. A. (1990). Numerical-analytical method for neural network investigation. In *Proceedings of The International Symposium on Neural Networks and Neural Computing - NEURONET'90* (pp. 217-222). Prague, Czechoslovakia.
Blair, A. D. (1997). Scaling-up RAAMs. (Technical Report CS-97-192). Brandeis University, Department of Computer Science.
Fedoseyeva, T. V. (1992). The problem of training neural network training to recognize word roots. In *Neuron-like networks and neurocomputers* (pp. 48-54). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).
Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6*, 205-254.
Feldman, J. A. (1989). Neural Representation of Conceptual Knowledge. In L. Nadel, L. A. Cooper, P. Culicover, & R. M. Harnish (Eds.), *Neural connections, mental computation* (pp. 68-103). Cambridge, Massachusetts, London, England: A Bradford Book, The MIT Press.
Foldiak, P., & Young, M. P. (1995). Sparse Coding in the Primate Cortex. In M. A. Arbib (Ed.), *Handbook of brain theory and neural networks* (pp. 895-898). Cambridge, MA: MIT Press.
Frasconi, P., Gori, M., & Sperduti, A. (1997). A general framework for adaptive processing of data structures. *Technical Report DSI-RT-15/97.* Firenze, Italy: Universita degli Studi di Firenze, Dipartimento di Sistemi e Informatica.
Frolov, A. A., & Muraviev, I. P. (1987). *Neural models of associative memory*. Moscow: Nauka. (In Russian).
Frolov, A. A., & Muraviev, I. P. (1988). Informational characteristics of neuronal and synaptic plasticity. *Biophysics, 33*, 708-715.
Frolov, A. A. (1989). Information properties of bilayer neuron nets with binary plastic synapses. *Biophysics, 34*, 868-876.
Gayler, R. W. (1998). Multiplicative binding, representation operators, and analogy. In K. Holyoak, D. Gentner, & B. Kokinov (Eds.), *Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences*. (p. 405). Sofia, Bulgaria: New Bulgarian University. (Poster abstract. Full poster available at: http://cogprints.soton.ac.uk/abs/comp/199807020).
Halford, G. S., Wilson W. H., & Phillips S. (in press). Processing capacity defined by relational complexity: implications for comparative, developmental, and cognitive psychology. *Behavioral and Brain Sciences.*
Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory* (pp. 161-187). Hillside, NJ: Lawrence Erlbaum Associates.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence, 46*, 47-76.

Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1986). Distributed representations. In D. E. Rumelhart, J. L. McClelland, & the PDP research group (Eds.), *Parallel distributed processing: Exploration in the microstructure of cognition 1: Foundations* (pp. 77-109). Cambridge, MA: MIT Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA, 79*, 2554-2558.

Hopfield, J. J., Feinstein, D. I., & Palmer, R. G. (1983). "Unlearning" has a stabilizing effect in collective memories. *Nature, 304*, 158-159.

Hummel, J. E., & Holyoak K. J. (1997). Distributed representations of structure: A theory of analogical access and mapping. *Psychological Review, 104*, 427-466.

Kanerva, P. (1988). *Sparse distributed memory*. Cambridge, MA: MIT Press.

Kanerva, P. (1994). The Spatter Code for encoding concepts at many levels. In M. Marinaro and P.G. Morasso (eds.), *ICANN '94, Proceedings of International Conference on Artificial Neural Networks* (Sorrento, Italy), *1*, pp. 226-229. London: Springer-Verlag.

Kanerva, P. (1996). Binary Spatter-Coding of Ordered K-tuples. In C. von der Malsburg, W. von Seelen, J. C. Vorbruggen, & B. Sendhoff (Eds.). *Proceedings of the International Conference on Artificial Neural Networks - ICANN'96, Bochum, Germany. Lecture Notes in Computer Science, 1112,* 869-873. Berlin: Springer.

Kanerva, P. (1998). Encoding structure in Boolean space. In L. Niklasson, M. Boden, and T. Ziemke (eds.), *ICANN 98: Perspectives in Neural Computing* (Proceedings of the 8th International Conference on Artificial Neural Networks, Skoevde, Sweden), *1*, pp. 387-392. London: Springer.

Kasatkin, A. M., & Kasatkina, L. M. (1991). A neural network expert system. In *Neuron-like networks and neurocomputers* (pp. 18-24). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M. (1980). *Tools and techniques for development of neuron-like networks for robot control*. Unpublished Dr. Sci. dissertation. Kiev, Ukrainian SSR: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M. (1988). Elements of stochastic neuron-like network theory. In *Internal Report "Kareta-UN"* (pp. 10-95). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M. (1992) *Associative neuron-like structures*. Kiev: Naukova Dumka. (In Russian).

Kussul, E. M. (1993). On some results and prospects of development of associative-projective neurocomputers. In *Neuron-like networks and neurocomputers* (pp. 4-11). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M., & Baidyk, T. N. (1990). Design of a neural-like network architecture for recognition of object shapes in images. *Soviet Journal of Automation and Information Sciences, 23(5),* 53-58.

Kussul, E. M., & Baidyk, T. N. (1993). On information encoding in associative-projective neural networks. (Preprint 93-3). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M., Baidyk, T. N., Lukovich, V. V., & Rachkovskij, D. A. (1993). Adaptive neural network classifier with multifloat input coding. In *Proceedings of NeuroNimes'93, Nimes, France, Oct. 25-29, 1993*. EC2-publishing.

Kussul, E. M., & Kasatkina, L. M. (1999). Neural network system for continuous handwritten words recognition. (Submitted).

Kussul, E. M., & Rachkovskij, D. A. (1991). Multilevel assembly neural architecture and processing of sequences. In A .V. Holden & V. I. Kryukov (Eds.), *Neurocomputers and Attention: Vol. II. Connectionism and neurocomputers* (pp. 577-590). Manchester and New York: Manchester University Press.

Kussul, E. M., Rachkovskij, D. A., & Baidyk, T. N. (1991a). Associative-Projective Neural Networks: architecture, implementation, applications. In *Proceedings of the Fourth International Conference "Neural Networks & their Applications", Nimes, France, Nov. 4-8, 1991* (pp. 463-476).

Kussul, E. M., Rachkovskij, D. A., & Baidyk, T. N. (1991b). On image texture recognition by associative-projective neurocomputer. In C. H. Dagli, S. Kumara, & Y. C. Shin (Eds.), *Proceedings of the ANNIE'91 conference "Intelligent engineering systems through artificial neural networks"* (pp. 453-458). ASME Press.

Lansner, A., & Ekeberg, O. (1985). Reliability and speed of recall in an associative network. *IEEE Trans. Pattern Analysis and Machine Intelligence, 7*, 490-498.

Lavrenyuk, A. N. (1995). Application of neural networks for recognition of handwriting in drawings. In *Neurocomputing: issues of theory and practice* (pp. 24-31). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Legendy, C. R. (1970). The brain and its information trapping device. In Rose J. (Ed.), *Progress in cybernetics, vol. I.* New York: Gordon and Breach.

Marr, D. (1969). A theory of cerebellar cortex. Journal of Physiology, *202*, 437-470.

Milner, P. M. (1974). A model for visual shape recognition. *Psychological Review, 81*, 521-535.

Milner, P.M. (1996). Neural representations: some old problems revisited. *Journal of Cognitive Neuroscience, 8*, 69-77.

Palm, G. (1980). On associative memory. *Biological Cybernetics, 36*, 19-31.

Palm, G. (1993). The PAN system and the WINA project. In P. Spies (Ed.), *Euro-Arch'93* (pp. 142-156). Springer-Verlag

Palm, G. & Bonhoeffer, T. (1984). Parallel processing for associative and neuronal networks. *Biological Cybernetics, 51*, 201-204.

Plate, T. A. (1991). Holographic Reduced Representations: Convolution algebra for compositional distributed representations. In J. Mylopoulos & R. Reiter (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 30-35). San Mateo, CA: Morgan Kaufmann.

Plate, T. A. (1994). Distributed representations and nested compositional structure. PhD Thesis, Department of Computer Science, University of Toronto.

Plate, T. A. (1995). Holographic Reduced Representations. *IEEE Transactions on Neural Networks, 6*, 623-641.

Plate, T. (1997). A common framework for distributed representation schemes for compositional structure. In F. Maire, R. Hayward, & J. Diederich (Eds.), *Connectionist systems for knowledge representation and deduction* (pp. 15-34). Queensland University of Technology.

Plate, T.A. (to appear). Structured Operations with Vector Representations. Expert Systems: The International Journal of Knowledge Engineering and Neural Networks, Special Issue on Connectionist Symbol Processing.

Plate, T. (submitted). Randomly connected sigma-pi neurons can form associative memories.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence, 46*, 77-105.

Rachkovskij, D. A. (1990a). On numerical-analytical investigation of neural network characteristics. In *Neuron-like networks and neurocomputers* (pp. 13-23). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Rachkovskij, D. A. (1990b). *Development and investigation of multilevel assembly neural networks*. Unpublished Ph.D. dissertation. Kiev, Ukrainian SSR: V. M. Glushkov Institute of Cybernetics. (In Russian).

Rachkovskij, D. A. (1996). Application of stochastic assembly neural networks in the problem of interesting text selection. In *Neural network systems for information processing (*pp. 52-64). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Rachkovskij, D. A. (to appear). Representation and Processing of Structures with Binary Sparse Distributed Codes. IEEE Transactions on Knowledge and Data Engineering, Special Issue on Connectionist Models for Learning In Structured Domains.

Rachkovskij, D. A. & Fedoseyeva, T. V. (1990). On audio signals recognition by multilevel neural network. In *Proceedings of The International Symposium on Neural Networks and Neural Computing - NEURONET'90* (pp. 281-283). Prague, Czechoslovakia.

Rachkovskij, D. A. & Fedoseyeva T. V. (1991). Hardware and software neurocomputer system for recognition of acoustical signals. In *Neuron-like networks and neurocomputers* (pp. 62-68). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Shastri, L. & Ajjanagadde, V. (1993). From simple associations to systematic reasoning: connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences, 16*, 417-494.

Sjödin, G. (1998). The Sparchunk Code: a method to build higher-level structures in a sparsely encoded SDM. In *Proceedings of IJCNN'98* (pp. 1410-1415), IEEE, Piscataway, NJ: IEEE.

Sjödin, G., Kanerva, P., Levin, B., & Kristoferson, J. (1998). Holistic higher-level structure-forming algorithms. In *Proceedings of 1998 Real World Computing Symposium - RWC'98* (pp. 299-304).

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence, 46*, 159-216.

Sperduti, A. (1994). Labeling RAAM. *Connection Science, 6*, 429-459.

Sperduti, A. & Starita, A. (1997). supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks, 8*, 714-735.

Tsodyks, M. V. (1989). Associative memory in neural networks with the Hebbian learning rule. *Modern Physics Letters B, 3*, 555-560.

Touretzky, D. S. (1990). BoltzCONS: Dynamic symbol structures in a connectionist network. *Artificial Intelligence, 46*, 5-46.

Touretzky, D. S. (1995). Connectionist and symbolic representations. In M. A. Arbib (Ed.), *Handbook of brain theory and neural networks* (pp. 243-247). Cambridge, MA: MIT Press.

Touretzky, D. S., & Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science, 12*, 423-466.

Vedenov, A. A. (1987). "Spurious memory" in model neural networks. (Preprint IAE-4395/1). Moscow: I. V. Kurchatov Institute of Atomic Energy.

Vedenov, A. A. (1988). *Modeling of thinking elements*. Moscow: Science. (In Russian).

von der Malsburg, C. (1981). The correlation theory of brain function. (Internal Report 81-2). Gottingen, Germany: Max-Planck-Institute for Biophysical Chemistry, Department of Neurobiology.

von der Malsburg, C. (1985). Nervous structures with dynamical links. *Ber. Bunsenges. Phys. Chem., 89*, 703-710.

von der Malsburg, C. (1986) Am I thinking assemblies? In G. Palm & A. Aertsen (Eds.), *Proceedings of the 1984 Trieste Meeting on Brain Theory* (pp. 161-176). Heidelberg: Springer-Verlag.

Willshaw, D. (1981). Holography, associative memory, and inductive generalization. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory* (pp. 83-104). Hillside, NJ: Lawrence Erlbaum Associates.

Willshaw, D. J., Buneman, O. P., & Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature, 222*, 960-962.

Figure captions

Figure 1. Growth of the density p' of 1s in the composite codevectors of higher hierarchical levels (see equations 2.1 and 2.2). Here each codevector of the higher level is formed by bit disjunction of S codevectors of the preceding level. Items at each level are uncorrelated. The codevectors of base-level items are independent with the density of 1s equal to p. The number of hierarchical levels is L. (For any given number of base-level items, the total number of 1s in the composite codevectors is obviously limited by the number of 1s in the disjunction of base-level codevectors).

Figure 2. Hatched circles represent patterns of active units encoding items; formed connections are plotted by arrowed lines.(A) Formation of a false assembly. When three assemblies (**abd**; **bce**; **acf**) are consecutively formed in a neural network by connecting all active units of patterns encoding their items, the fourth assembly **abc** (gridy hatch) is formed as well, though its pattern was not explicitly presented to the network. (B) Preventing of a false assembly. If each of three assemblies is formed by connecting only subsets of active units encoding the component items, then the connectivity of the false assembly is weak. $\mathbf{x_{yz}}$ denotes the subset of units encoding item x when it is combined with items y and z.. The pairwise intersections of the small circles represent the false assembly.

Figure 3. A neural-network implementation of permutive conjunctive thinning. The same N-dimensional binary pattern is activated in the input neural fields $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{in2}}$. It is a superposition of several component codevectors. $\mathbf{f}_{\text{in1}}$ is connected to $\mathbf{f}_{\text{out}}$ by a bundle of direct projective connections. $\mathbf{f}_{\text{in2}}$ is connected to $\mathbf{f}_{\text{out}}$ by a bundle of permutive connections. Conjunction of the superimposed component codevectors and their permutation is obtained in the output neural field $\mathbf{f}_{\text{out}}$, where the neural threshold θ=1.5.

Figure 4. (A), (B). Algorithmic implementations of the additive version of the Context- Dependent Thinning procedure. Parameter *seed* defines a configuration of shift permutations. For small K, checking that r is unique would be useful.

Figure 5. A neural-network implementation of the additive version of Context-Dependent Thinning procedure. There are four neural fields with the same number of neurons: two input fields $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{in2}}$, the output field $\mathbf{f}_{\text{out}}$, the intermediate field $\mathbf{f}_{\text{int}}$. The neurons of $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{out}}$ are connected by the bundle of direct projective connections (1-to-1). $\mathbf{f}_{\text{int}}$ and $\mathbf{f}_{\text{out}}$ are also connected in the same manner. The same binary pattern $\mathbf{z}$ (corresponding to superimposed component codevectors) is in the input fields $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{in2}}$. The intermediate field $\mathbf{f}_{\text{int}}$ is connected to the input field $\mathbf{f}_{\text{in2}}$ by K bundles of permutive projective connections. The number K of required bundles is estimated in Table 3. Only two bundles are shown here: one by solid lines and one by dotted lines. The threshold of $\mathbf{f}_{\text{int}}$ neurons is 0.5. Therefore $\mathbf{f}_{\text{int}}$ accumulates (by bit disjunction) various permutations of the pattern $\mathbf{z}$ in $\mathbf{f}_{\text{in2}}$. The threshold of $\mathbf{f}_{\text{out}}$ is equal to 1.5. Hence this field performs conjunction of the pattern $\mathbf{z}$ from $\mathbf{f}_{\text{in1}}$ and the pattern of K permuted and superimposed $\mathbf{z}$ from $\mathbf{f}_{\text{int}}$. $\mathbf{z}$, $\langle\mathbf{z}\rangle$, $\mathbf{w}$ correspond to the notation of Figure 4.

Figure 6. A neural-network implementation of the subtractive Context-Dependent Thinning procedure. There are three neuron fields of the same number of neurons: two input fields $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{in2}}$, as well as the output field $\mathbf{f}_{\text{out}}$. The copy of the input vector $\mathbf{z}$ is in both input fields. The neurons of $\mathbf{f}_{\text{in1}}$ and $\mathbf{f}_{\text{out}}$ are connected by the bundle of direct projective connections (1-to-1). The neurons of $\mathbf{f}_{\text{in2}}$ and $\mathbf{f}_{\text{out}}$ are connected by K bundles of independent permutive connections. (Only two bundles of permutive connections are shown here: one by solid lines, and one by dotted lines). Unlike Figure 5, the synapses of permutive connections are inhibitory (the weight is -1). The threshold of the output field neurons is 0.5. Therefore the neurons of $\mathbf{z}$ remaining active in $\mathbf{f}_{\text{out}}$ are those for which none of the permutive connections coming from $\mathbf{z}$ are active. As follows from Table 4, K is approximately the same for the number S = 2,...,5 of component codevectors of certain density p.

Figure 7: (A) Overlap of thinned composite codevectors and (B) Overlap of component subsets in thinned composite codevectors - versus the overlap of the corresponding unthinned composite codevectors for various versions of thinning procedures. CDTadd - the additive, CDTsub - the subtractive, CDTadd-sl - the self-exclusive additive, CDTsub-sl - the self-exclusive subtractive CDT procedure, Paired-M - permutive conjunctive thinning; the densities of component codevectors are chosen to obtain M of 1s in the thinned codevector. For component codevectors, N=100000, M≈1000. The number of component codevectors: 5. The results are averaged over 50 runs with different random codevectors.

Figure 8: (A) Overlap of thinned composite codevectors and (B) Overlap of component subsets in thinned composite codevectors for various "depth" of additive (CDTadd) and subtractive (CDTsub) procedures of Context-Dependent Thinning. There are five components in the composite item. Therefore the input composite codevector includes approximately 5M of 1s. The composite codevector is thinned to have from 4M to M/4 of 1s. Two curves for thinning depth M are consistent with the corresponding curves in Figure 7. For all component codevectors, N=100000, M≈1000. The results are averaged over 50 runs with different random codevectors.

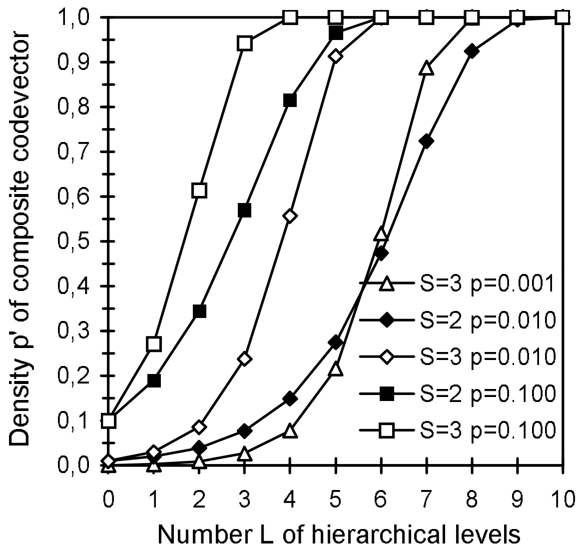

Figure 1.

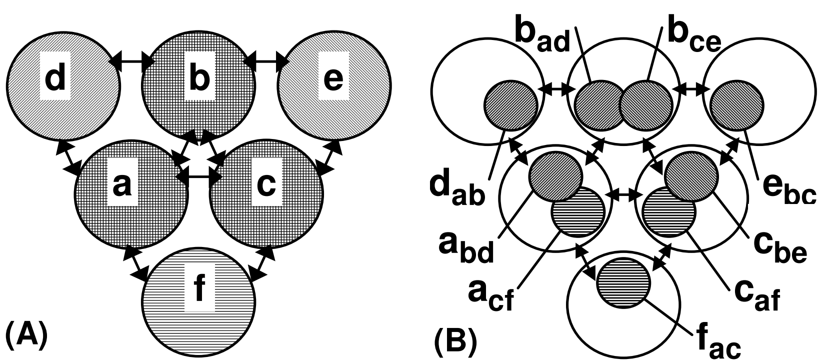

Figure 2.

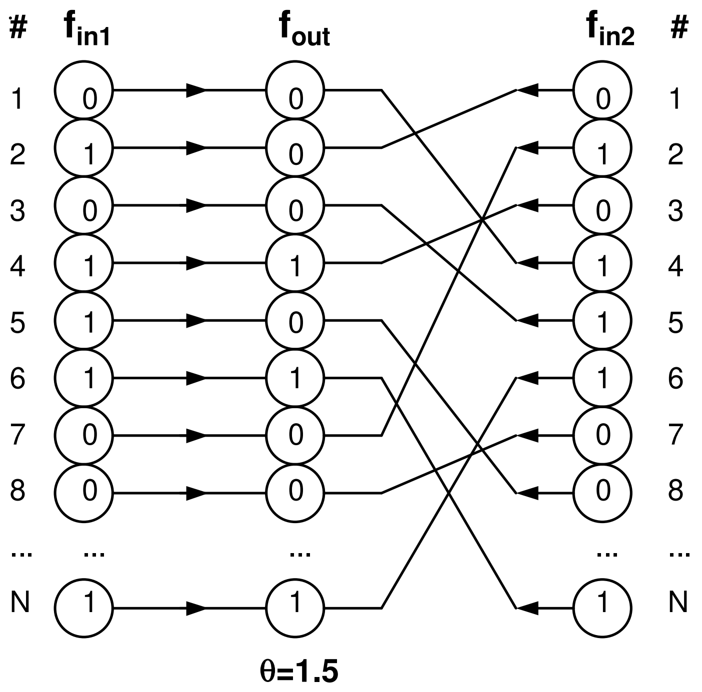

Figure 3.

```
The input codevector z = x1 ∨ x2 ∨ ... ∨ xS                (A)
The output (thinned) codevector is ⟨z⟩
Calculate K=ln(1-1/S)/ln(1-pS)
Set auxiliary vector w to 0
Seed the random-number generator: randomize(seed).

for (k=1, 2,..., K)
        if ((r=rand()) ≠ 0)
                for (i=1, 2,..., N)
                        wi = wi ∨ zi+r modulo N
for (i=1, 2,..., N)
        ⟨z⟩i = zi ∧ wi
```

```
The input codevector z = x1 ∨ x2 ∨ ... ∨ xS                (B)
Set the output (thinned) codevector ⟨z⟩ to 0.
Seed the random-number generator: randomize(seed).

while(|⟨z⟩| < M)
        if ((r=rand()) ≠ 0)
                for (i=1, 2,..., N)
                        ⟨z⟩i = ⟨z⟩i ∨ (zi ∧ zi+r modulo N)
```

Figure 4. (A), (B).

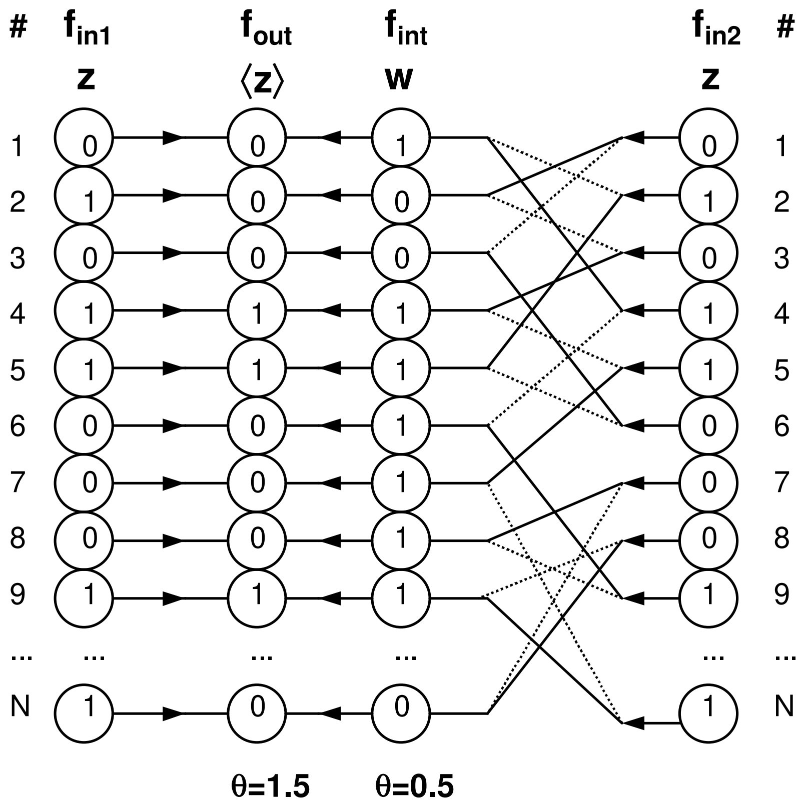

Figure 5.

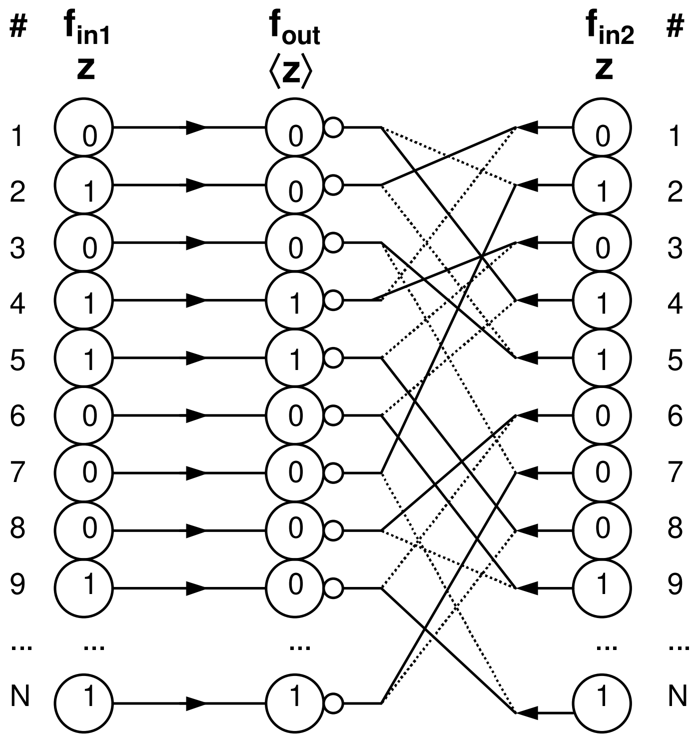

Figure 6.

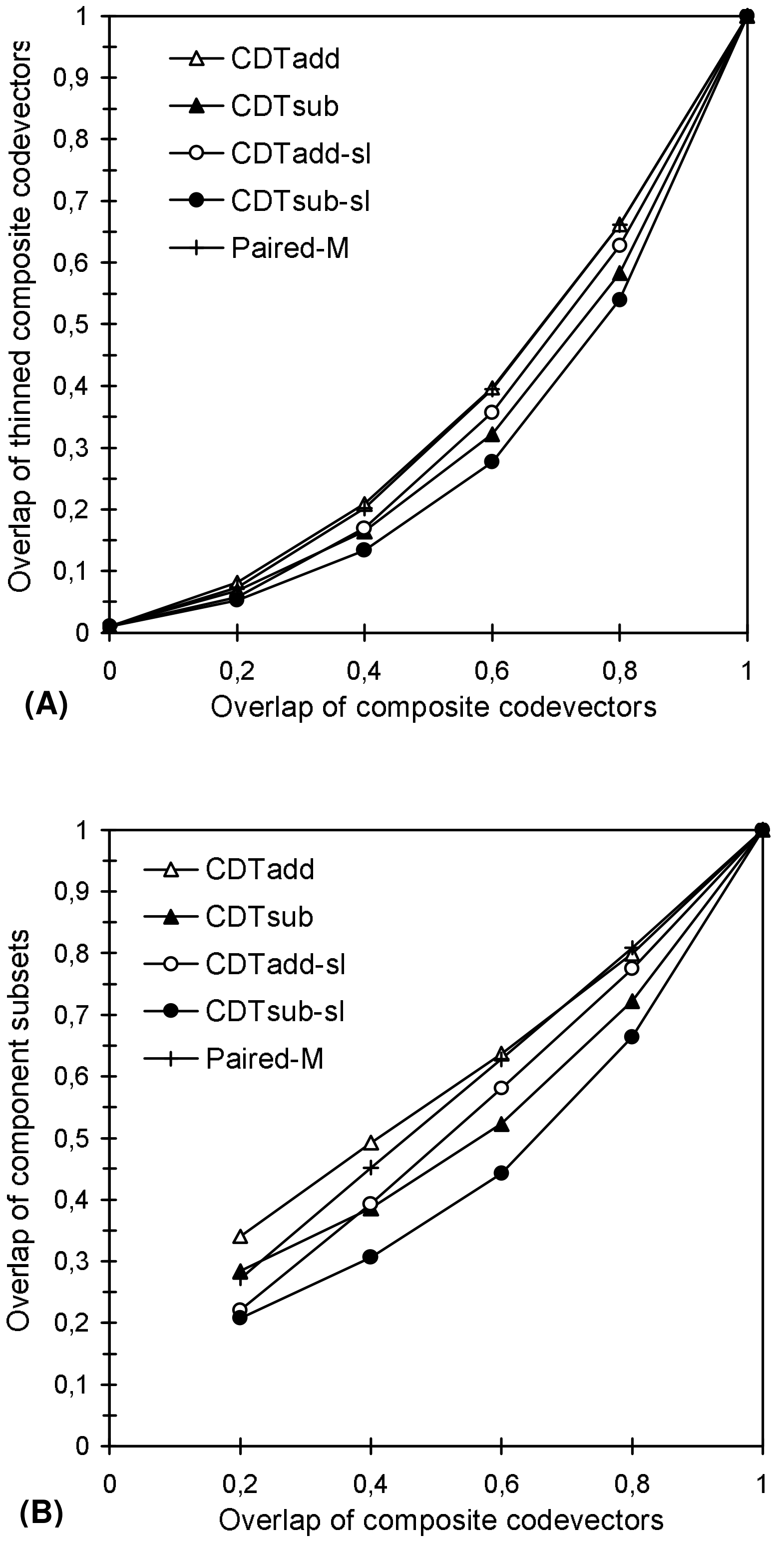
CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Overlap of thinned composite codevectors
Overlap of composite codevectors
(A)
CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Overlap of component subsets
Overlap of composite codevectors
(B)

Figure 7.

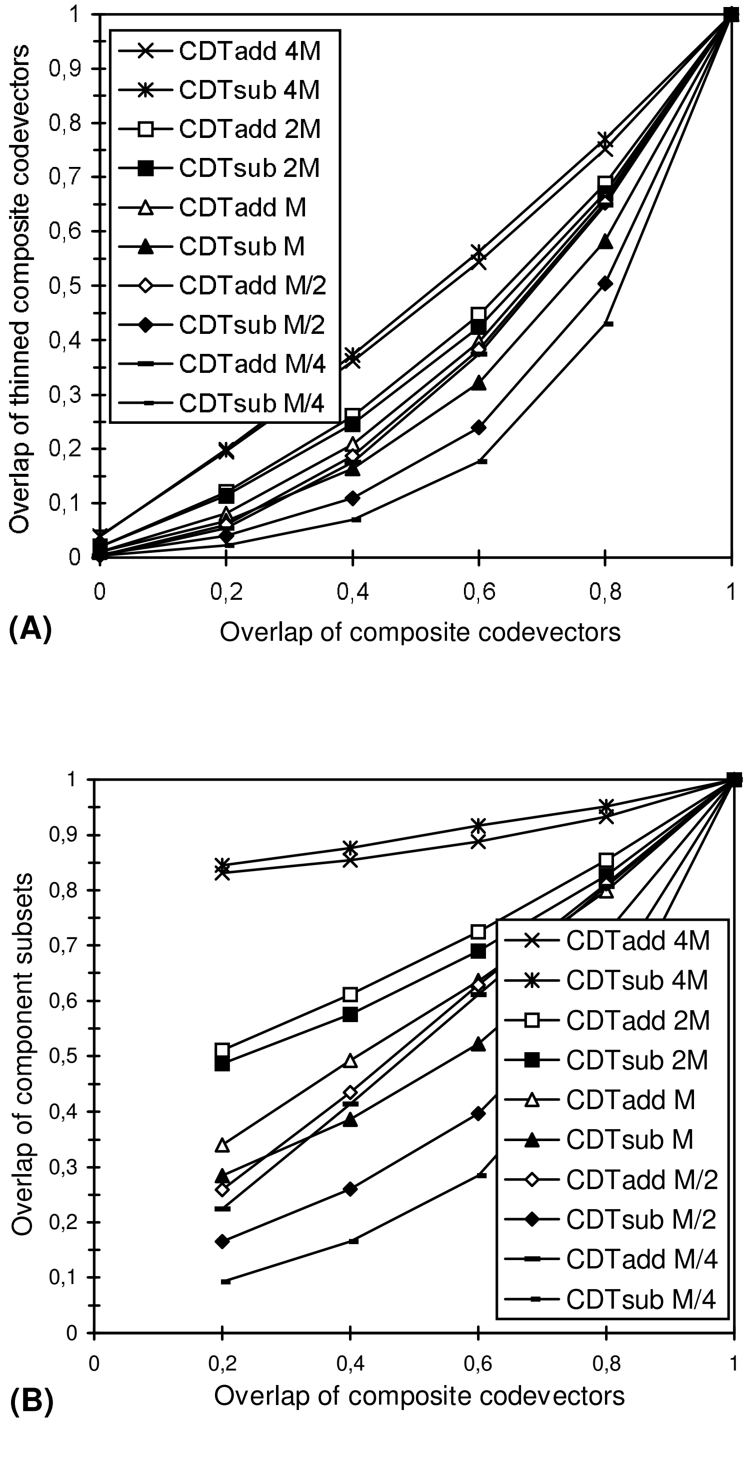
Overlap of thinned composite codevectors
Overlap of composite codevectors
CDTadd 4M
CDTsub 4M
CDTadd 2M
CDTsub 2M
CDTadd M
CDTsub M
CDTadd M/2
CDTsub M/2
CDTadd M/4
CDTsub M/4
(A)
Overlap of component subsets
Overlap of composite codevectors
(B)

Figure 8.