Representation and processing of structures with binary sparse distributed codes

Dmitri A. Rachkovskij
V. M. Glushkov Cybernetics Center
Pr. Acad. Glushkova 40
Kiev 22, 252022
Ukraine
dar@infrm.kiev.ua

Paper draft: 26 pages. Date: June 30, 1999.

Keywords: sparse coding, binary coding, binding, representation of structure, hierarchical representation, nested representation, long-term memory, analogy, compositional distributed representations, connectionist symbol processing, analogical retrieval, analogical mapping

Abstract

The schemes for compositional distributed representations include those allowing on-the-fly construction of fixed dimensionality codevectors to encode structures of various complexity. Similarity of such codevectors takes into account both structural and semantic similarity of represented structures. In this paper we provide a comparative description of sparse binary distributed representation developed in the frames of the Associative-Projective Neural Network architecture and more well-known Holographic Reduced Representations of Plate and Binary Spatter Codes of Kanerva.

The key procedure in Associative-Projective Neural Networks is Context-Dependent Thinning which binds codevectors and maintains their sparseness. The codevectors are stored in structured memory array which can be realized as distributed auto-associative memory. Examples of distributed representation of structured data are given. Fast estimation of similarity of analogical episodes by the overlap of their codevectors is used in modeling of analogical reasoning for retrieval of analogs from memory and for analogical mapping.

.

1. Introduction

The problem of representation is of great importance in Artificial Intelligence (AI). In local connectionist models each item is represented by a single unit, node, or neuron (at least in long-term memory). Such representations are criticized in respect to their limited information capacity and alienation from the semantic base.

For example, in order to represent nested compositional structures it is necessary to represent their constituents, various compositions of those constituents, combinations of those compositions, etc. For significant number of complex recursive structures, the number of possible combinations and localist elements required to represent them becomes prohibitive because of exponential growth.

At the same time, a local representational element (or symbol) does not carry immediate information about its constituents and acts just as a meaningless pointer. For example, in order to estimate the similarity of hierarchical structures encoded with localist representations, it is generally necessary to find and match their constituents down to the lowest hierarchical level using pointer following.

In fully distributed representations, each item is encoded by a pattern of activity over the pool of units. The number of different non-orthogonal patterns exponentially exceeds the dimensionality of the coding pool. It allows for a high information capacity of distributed representations. An estimation of item similarity can be realized by the dot-product of the vectors corresponding to their activity patterns.

Up to now distribution representations are sometimes perceived as unsuitable for hierarchical (recursive, nested) structure representation. However in fact a number of schemes has been proposed for distributed representation of structured data, including BoltzCONS of Touretzky [55], Reduced Descriptions of Hinton [20], Tensor Products of Smolensky [51], RAAMs of Pollack [43], LRAAMs of Sperduti [52] (see also [41] for a review). Of special interest to us are the schemes where the codes representing new structures of various complexity are constructed on-the-fly (without special learning) and have the same dimensionality. Associative-Projective Neural Networks (APNNs) [29], Holographic Reduced Representations (HRRs) [40], and Binary Spatter Codes (BSCs) [26] are among such schemes.

The feasibility to estimate easily the similarity of complex structured information representations is essential for solution of many AI problems [29]. One of these problems is modeling of human analogy-making. It requires taking into account structure and semantics of analogs. Due to the common supposition that distributed representations poorly deal with structure, it is natural that most influential computational models of analogical reasoning (ARCS-ACME [53], [22], MAC/FAC-SME [9], [7], Copycat [21]) are essentially based on symbolic or localist representations.

However the quest to enhance the semantic basis of analog representations and to reach a higher degree of neurological relevancy leads to the attempts to augment the models of analogy with some share of distributed representations (LISA [24], STAR2 [16], DRAMA [6]). Operations on exclusively distributed representations useful for analogical processing were demonstrated by Plate [39], [42] using HRRs and Kanerva [27], [28] using BSCs.

In this paper an approach to representation and manipulation of structures using binary sparse distributed codes and techniques of APNNs is considered and compared with the approaches adopted in HRRs and BSCs. In section 2 we provide a comparative description of basic features and operations of APNNs, HRRs, and BSCs. Various types of distributed representation of structures are presented is section 3. Some considerations concerning organization and implementation of long-term memory for structures are given in section 4. In section 5 we apply APNN-style representations for retrieval and mapping of toy analogical episodes and compare with the results of Plate. Sections 6 and 7 are discussion and conclusion correspondingly.

2. On-the-fly distributed representations of structure: basic features and operations

In order to represent nested compositional structures efficiently, the following features of APNN-style representations are essential:

- Representations should be fully distributed. An item of any complexity or nesting level, including the lowest (basic) level items, is represented by a distributed activity pattern over the pool of units. It allows to avoid the limitation of the number of representable items by the number of available units.
- Dimensionality should be preserved. The dimensionality of the coding units' pool for items of any complexity or nesting level is the same. It prevents the problems concerning the growth of composite items' code dimensionality.

- Representations should be constructed "on-the-fly". The code of a composite item of any complexity or nesting level is formed from the codes of its constituents without training. It allows fast recursive construction of representations of arbitrary items.
- Similar items should produce similar representations. It allows to estimate the similarity of items by the dot product of their codevectors. The degree of similarity varies gradually.

These requirements are met, apart from APNNs, by HRRs and by BSCs. Therefore in this section we consider the commonalities and the differences between these schemes. For comparison of APNNs with other distributed schemes for structure representation, see [46].

2.1. The nature of distributed representations

In fully distributed representations each item (a feature, an object, a relation, a structure, etc.) is encoded by an activity pattern over the total pool of units. It is convenient to represent the pool of N units by N-dimensional vector, and the activity of each unit by the value of the corresponding vector element. Let us denote codevectors by a bold-face font, and the items by a cursive font.

In HRRs, elements of codevectors are real numbers with the Gaussian distribution of mean 0 and variance 1/N. For example, N=2048 [42]. Similarity of the HRR codevectors is calculated as their dot product.

The BSCs are binary codevectors (or "codewords" in terms of Kanerva [26]). Each vector element can be 0 or 1 with the probability 0.5. Let us denote the number of 1s in a codevector by M. The density of a codevector **X** can be defined as the probability of 1s in it: p(**X**)=M/N. Therefore the codevectors of BSCs are "dense": p(**X**)=0.5. For example, N=10000, M=5000 [27]. The distance between two codevectors is defined as the Hamming distance. Calculation of codevectors' similarity by the dot product actually counts the number of coinciding 1s.

Codevectors in the APNN scheme are sparse binary vectors: p(**X**)<<1.0. It is closer to neurologic data and useful for associative memory capacity. For example, p(**X**)=0.01, N=100000, M=1000.

Usually the elements of codevectors corresponding to dissimilar items of the lowest (base) hierarchical level are generated randomly and independently. Generated codevectors are stored. Encoding of composite structures will be considered below.

2.2. Superposition

A set of items is represented by superposition of their codevectors. In HRRs, superposition is realized by elementwise addition. In BSCs, it is realized by bitwise addition of codevectors (with subsequent thresholding for binarization). In APNNs, codevector superposition is implemented by bitwise disjunction. Obviously, for all these types of superposition the result preserves the dimensionality of initial codevectors.

Superposition also preserves similarity of the result with each of superimposed codevectors. For independent codevectors, **D=A+B** is similar to **A** and **B** ('+' denotes superposition). This type of similarity is called "unstructured" similarity [41]. The similarity of codevectors to their superposition decreases as the number of superimposed codevectors increases. Therefore superposition similarity can be also considered as "additive".

Unstructured similarity allows to determine which codevectors are superimposed by picking the codevectors (from their total alphabet) that are most similar to the superimposed codevector.

2.3. Binding

2.3.1. Necessity of binding and some related problems

Superposition alone is not enough to encode structure. Under superposition, the components of structure are mixed together, and the information on their combination or order in substructures is not preserved. Composition by superposition was the reason to blame distributed representations for the lack of structure. Two common formulations of this problem are "ghosts" and "superposition catastrophe" ([8], [57], see also [46]).

The simplest illustration could be as follows. Let there be component items *A*,*B*,*C* and composite items *AB*, *AC*, *BC* which are encoded by the simultaneous “all-or-none” activation of component item representations (Figure 1). If representations of any two of three composite items (e.g. *AB* and *BC*) are activated, the third composite item will become active as well, though unbidden (here, "ghost" *AC*). In "superposition catastrophe" formulation, the same situation is described as follows: which two composite items are really present, if representations of component items *A*,*B*,*C* are activated? A relevant problem is that it is impossible to encode the order or sequence of items by their simultaneous activation: pattern for *ABC* is indistinguishable from *CBA* or *BCA*.

It should be noted that these problems persist both for localist and distributed representations of the component items *A*,*B*,*C*, if the presence of a composite item is encoded by simultaneous activation of its component representations. These problems also manifest themselves as "spurious memories" ([23], [56]) for superposition learning rules (e.g. Hebb’s learning rule [19] for matrix-type distributed associative memory)

Thus in order to represent compositional structures, one should be able to represent "bindings" of component items. In localist models, such bindings are represented by introduction or allocation of new units (e.g., the *AB* unit to represent the binding of *A* and *B*). It increases the dimensionality of units' pool when new bindings and structures are introduced (Figure 2).

A local node itself does not carry any information concerning the composite item it represents. Therefore between-unit connections are needed to point to the nodes representing its constituents. Thus in order to estimate the mere similarity of the base-level components of two structures represented by local nodes, both of them should be unfolded through the nodes representing their base-level components (Figure 3).

If the comparison should take into account the structural arrangement of components, much more complicated techniques are needed, as exemplified by the models of analogy processing, where analogs are structured propositions (Figure 4). These techniques include construction and settling of special constraint satisfaction networks which find the best match between the corresponding items of analogs (ACME [22], DRAMA [6]), or symbolic processing of virtual networks representing those correspondences (SME [7]), or use of "mapping connections" between "dynamic bindings" ([36], [48]) represented by temporal co-activation of units in existing localist network (LISA [24]).

In the distributed representations of structure we consider here, binding of codevectors is implemented by special operations which do not change dimensionality. It is important for representation of nested structures, where bindings of the lower hierarchical levels are used as the components of higher-level bindings.

2.3.2. Binding in HRRs and BSCs.

Circular convolution is used in HRRs to obtain the codevector **Z** which represents the binding of two codevectors **X** and **Y**:

$$Z_i = \sum_{j=1}^{N} X_j Y_{i-j}, \quad (1)$$

where subscripts are taken modulo-N. This operation can be also written as **Z=X∗Y**, where '∗' denotes circular convolution. Binding by circular convolution preserves the Gaussian distribution of codevector elements.

Unlike the result of superposition, the result **Z** of binding obtained by circular convolution is generally dissimilar to the codevectors **X** and **Y**. However another kind of similarity is preserved. **D=A∗B** is similar to **D'=A'∗B'**, if **A** is similar to **A'**, **B** is similar to **B'**. This kind of similarity is called "structured" similarity [41]. Binding by circular convolution preserves structured similarity in a multiplicative fashion. If **A** and **B** are independent, as well as **A'** and **B'**, then the similarity of **D** and **D'** is equal to the product of the similarities of **A** and **A'**, **B** and **B'**. E.g. if **A=A'**, the similarity of **D** and **D'** will be equal to the similarity of **B** and **B'**. If **A** is not similar to **A'**, **D** will have no similarity with **D'** irrespective of the similarity between **B** and **B'**.

This type of similarity is different from the structured similarity of superimposed codevectors: **D=A+B** will still be similar to **E=A+C** if **B** is dissimilar to **C**.

Gayler [12] proposed to use elementwise multiplication to bind real-valued codevectors. Binding in BSCs is implemented by elementwise XOR of codevectors. These binding operations preserve the similarity in approximately the same manner as circular convolution.

2.3.3. Binding in APNNs

In APNNs binding is realized by the Context-Dependent Thinning (CDT) procedure. The first version of this procedure was proposed by Kussul under the name "normalization" (e.g. [30], [33]). Various versions of the CDT (thinning) procedure, their algorithmic and neural network implementations are discussed in [46].

Here we will describe the "additive" version of the auto-CDT procedure [46]. In this version, single codevector is input to the procedure. This codevector is the disjunctive superposition (section 2.2) of two or several component codevectors to be bound:

$$\mathbf{Z} = \bigvee_{s=1}^{S} \mathbf{X}_s. \quad (2)$$

S is the number of components, $\mathbf{X}_s$ is the codevector of the s-th component. Then

$$\{\mathbf{Z}\} = \bigvee_{k=1}^{K} (\mathbf{Z} \wedge \tilde{\mathbf{Z}}(k)) = \mathbf{Z} \wedge \bigvee_{k=1}^{K} \tilde{\mathbf{Z}}(k). \quad (3).$$

Here $\{\mathbf{Z}\}$ is the thinned codevector, $\tilde{\mathbf{Z}}(k)$ is **Z** with permutated elements. Each k-th permutation is unique and fixed.

The CDT procedure reduces the density of the input vector down to some specified value. Usually it is comparable to the density of its component codevectors $\mathbf{X}_s$. The number K of permutated and disjunctively superimposed vectors is chosen so that the number of 1s in $\{\mathbf{Z}\}$ becomes as needed. For example, let the density of each component codevector be

$p(\mathbf{X}_s) = 0.01 \ \forall s,$

and let us bind 3 components (S=3). The density of the input superposition codevector is approximately

$p(\mathbf{Z}) \approx Sp(\mathbf{X}_s) = 3*0.01 = 0.03.$

$\mathbf{Z}^{\sim}$ and $\mathbf{Z}$ are independent because of permutation. Therefore the probability of 1s in their conjunction is

$p(\mathbf{Z} \wedge \mathbf{Z}^{\sim})=p(\mathbf{Z})p(\mathbf{Z}^{\sim})= 0.03*0.03=0.0009$.

Thus we need to superimpose by disjunction approximately 11 conjunctions of $\mathbf{Z}$ with its different unique permutations $\mathbf{Z}^{\sim}(k)$ in order to get

$p(\{\mathbf{Z}\}) \approx p(\mathbf{X}_s)$. That is, $K \approx 11$ in Eq.(3).

Unlike the binding operations in HRRs and BSCs, thinning in APNNs preserves both unstructured and structured similarity of the codes. Thinning preserves the 1s of $\mathbf{Z}$ (Eq. 3). Therefore the thinned codevector is similar to the codevectors of all components superimposed in the input codevector, thus preserving unstructured similarity.

The fraction of 1s in the output thinned codevector from each of component codevectors is proportional to the density of the component codevectors. For example, for independent $\mathbf{A},\mathbf{B},\mathbf{C}$ of the same density,

$\{\mathbf{Z}\} = \{\mathbf{A} \vee \mathbf{B} \vee \mathbf{C}\}$, $p(\{\mathbf{Z}\}\wedge\mathbf{A}) = p(\{\mathbf{Z}\}\wedge\mathbf{B}) = p(\{\mathbf{Z}\}\wedge\mathbf{C}) = p(\{\mathbf{Z}\})/3$.

If $p(\{\mathbf{Z}\})=p(\mathbf{A})=p(\mathbf{B})=p(\mathbf{C})$, approximately 0.33 of 1s will remain from each of the component codevectors.

However the subset of preserved 1s from each component depends on the composition of other components in the superimposed input codevector that was thinned. For the example of three-component composite items, the subset of 1s from $\mathbf{A}$ preserved in $\{\mathbf{A}\vee\mathbf{B}\vee\mathbf{C}\}$ is different from the subset of 1s preserved in $\{\mathbf{A}\vee\mathbf{B}\vee\mathbf{D}\}$ and $\{\mathbf{A}\vee\mathbf{D}\vee\mathbf{E}\}$ (Figure 5). Therefore thinned code is bound - representation of 1s from each component depends on the other components present.

Similar component sets produce similar thinned codes, preserving structured similarity. However, the character of structured similarity here is not multiplicative in terms of component codevectors, as for the previous binding schemes. In those schemes, single dissimilar component makes the binding dissimilar. For this procedure, due to the initial superposition, single similar component makes the output resulting codevectors similar.

The character of structured similarity is shown in Figure 6. The input vector was superposition of five independent component codevectors of the same density. The similarity of two input vectors was varied by changing the composition of components. For example, if we choose the first codevector as $\mathbf{Z1} = \mathbf{A} \vee \mathbf{B} \vee \mathbf{C} \vee \mathbf{D} \vee \mathbf{E}$ and the second one as $\mathbf{Z2} = \mathbf{A} \vee \mathbf{B} \vee \mathbf{C} \vee \mathbf{F} \vee \mathbf{H}$, their similarity (overlap) will be approximately 0.6.

Depending on the density of the thinned codevectors, their similarity varies from linear (additive similarity of superposition corresponding to the "shallow" thinning) to approximately quadratic. This character of similarity can be explained as follows. Overlap of thinned $\{\mathbf{Z1}\}$ and $\{\mathbf{Z2}\}$ (taking into account Eq. 3):

$$\{\mathbf{Z1}\}\wedge\{\mathbf{Z2}\} = \mathbf{Z1} \wedge \bigvee_{k=1}^{K1} \mathbf{Z1}^{\sim}(k) \wedge \mathbf{Z2} \wedge \bigvee_{k=1}^{K2} \mathbf{Z2}^{\sim}(k)$$

$$= \mathbf{Z1} \wedge \mathbf{Z2} \wedge \bigvee_{k=1}^{K1} \mathbf{Z1}^{\sim}(k) \wedge \bigvee_{k=1}^{K2} \mathbf{Z2}^{\sim}(k). \quad (4)$$

Since $\mathbf{Z1}$ and $\mathbf{Z2}$ have approximately the same density, let us assume K1=K2=K. Then

$$\bigvee_{k=1}^{K1} \mathbf{Z1}^{\sim}(k) \wedge \bigvee_{k=1}^{K2} \mathbf{Z2}^{\sim}(k) = \bigvee_{k=1}^{K} (\mathbf{Z1}^{\sim}(k) \wedge \mathbf{Z2}^{\sim}(k)) = \bigvee_{k=1}^{K} (\mathbf{Z1} \wedge \mathbf{Z2})^{\sim}(k).$$

Taking into account that $\mathbf{Z1} \wedge \mathbf{Z2}$ is independent from its permutations:

$$p(\{\mathbf{Z1}\} \wedge \{\mathbf{Z2}\}) = p(\mathbf{Z1} \wedge \mathbf{Z2})\, p\Big(\bigvee_{k=1}^{K} (\mathbf{Z1} \wedge \mathbf{Z2})^{\sim}(k)\Big). \quad (5)$$

Depending on K, and taking into account that permutation does not change the density of a vector, $p(\bigvee_{k=1}^{K} (\mathbf{Z1} \wedge \mathbf{Z2})^{\sim}(k))$ varies from $p(\mathbf{Z1} \wedge \mathbf{Z2})$ (K=1) to 1 (large K). Therefore

$$(p(\mathbf{Z1} \wedge \mathbf{Z2}))^2 \le p(\{\mathbf{Z1}\} \wedge \{\mathbf{Z2}\}) \le (p(\mathbf{Z1} \wedge \mathbf{Z2})). \quad (6)$$

The algorithm of the additive auto-CDT procedure is shown in Figure 7. A lot of orthogonal "patterns" of thinning (using different permutations) is possible depending on the seed in the algorithmic implementation. We will denote them by different types of brackets around the input vector: {**Z**}, [**Z**], (**Z**), **/Z/** , **Z**\, etc.

2.4. Unbinding

To retrieve component codevectors from their bindings, various techniques are used in different representation schemes.

Special unbinding operation is needed for HRRs and BSCs, because their bindings are not similar to the codevectors of bound components. To reconstruct **Y** from **Z**=**X**∗**Y**, **X** should be known. If **X** is known, its inverse can be found. According to Plate (e.g. [40], [41]), the approximate inverse of **X:** $\mathbf{X}^T$ is enough for unbinding and easy to obtain: $X_i^T = X_{N-i}$, where subscripts are modulo-N. Unbinding gives

$$\mathbf{X}^T * \mathbf{Z} = \mathbf{X}^T * \mathbf{X} * \mathbf{Y} = \mathbf{Y} + \text{noise}. \quad (7)$$

To reconstruct an accurate version of **Y**, the noisy version can be compared with all component codevectors to find the most close one.

In BSCs, unbinding is implemented by XOR.

In APNNs, the similarity of component codevectors with their reduced representation in the thinned code is preserved. Therefore in order to retrieve in full size the component codevectors using the thinned codevector, no intermediate unbinding operation is necessary - it is enough to find the most similar codevectors in the component memory. (Memory is discussed in section 4).

2.5. Normalization

The operation of normalization is used in HRRs to maintain the overall strength of codevectors and statistical distribution of their elements after operations implemented over them. The normalized codevector **X** is denoted as <**X**> and is calculated as $\langle \mathbf{X} \rangle = \mathbf{X}/|\mathbf{X}|$, where $|\mathbf{X}|$ is the Euclidean length of **X**.

In BSCs, normalization is implemented by thresholding of each element of superimposed codevector. It maintains the binary character of the resulting codevector and its density of 0.5.

In APNNs, the CDT procedure not only binds the codevectors of components, but also simultaneously implements normalization by controlling the density of the resulting codevectors.

3. Distributed representation of relational structures

Let us consider how operations and procedures discussed above can be used to represent relational structures.

3.1. HRRs and BSCs for relational structures

HRRs and BSCs both use role-filler bindings to represent relational structure (e.g. [42], [26]). The relational instance or proposition "Spot bit Jane" is represented by HRRs as:

P(bite) = <P_structure(bite) + P_components(bite)>, (8)

where

P_structure(bite) = bite_agt∗spot + bite_obj∗jane, (9)

P_components(bite) = bite + <spot + jane>. (10)

P_structure(bite) represents the order of entities Spot and Jane in their "bite" relationship. The codevectors **spot** and **jane** are fillers bound correspondingly with the codevectors **bite_agt** and **bite_obj** which represent biting agent and bitten object respectively.

P_components(bite) represents the entities involved and relation label **bite** (the latter could be replaced by **bite_agt + bite_obj**). It allows to preserve the unstructured similarity of **P(bite)** to its component codevectors (which is not preserved by bindings in **P_structure(bite)**)**.** It is needed both to support surface (superficial) similarity of representation to its constituents and to be able to reconstruct by unbinding which filler is bound with which role. For example of the latter, unstructured similarity allows to find the presence of **bite_agt**, **bite_obj**, **spot**, **jane** in **P(bite)** and then find corresponding fillers or roles:

$\mathbf{bite_agt}^{T}$**∗P(bite) = spot** + noise, (11)

or

$\mathbf{jane}^{T}$**∗P(bite) = bite_obj** + noise. (12)

To represent the higher-level propositions which use lower-level propositions as their components, in HRRs lower-level propositions are used as the fillers for some higher-level roles. For example, the representation of "Spot bit Jane, causing Jane to flee from Spot" looks as [42]:

P = < cause + <P(bite)+P(flee)> + cause_antc∗P_bite + cause_cnsq∗P_flee >, (13)

where **P(bite)** is as above, and **P(flee)** is formed analogously:

P(flee) = < flee + <spot+jane> + flee_agt∗jane + flee_obj∗spot >. (14)

3.2 Representation of relational structure in APNNs

Representation of relational structure in APNNs has some peculiarities compared to the HRR representations considered above. Two types of binding may be distinguished in the APNN representation. The first type is analogous to the role-filler bindings in HRRs and is used to represent the correspondence or order of items. The second type is used to bind together a set of component items comprising a composite item. This type of binding is not pronounced in HRRs.

3.2.1. Binding for representation of correspondence or order

One version of encoding the correspondence of a filler to a certain role is analogous to the role-filler bindings in HRRs. In APNNs it is realized by explicit binding of the role codevector with the filler codevector. If the role is just a "position" (1st, 2nd, 3rd, ...) of an item in a proposition, this scheme encodes the order of filler items (see also [45], [32], [46]). The binding can be implemented by some version of the auto-CDT or hetero-CDT procedure. (Hetero-thinning is a special kind of thinning where the codevectors to be thinned are separated [45], [32], [46]).

Role-filler bindings using the auto-CDT procedure are exemplified by {**bite_agt** ∨ **spot**}, {**bite_obj** ∨ **jane**}, where **bite_agt** corresponds to the "1st place role in *bite* proposition", and **bite_obj** corresponds to the "2nd place role in *bite* proposition".

Another version of order encoding consists in changing physical position of an item codevector depending on its position (or place or ordinal number) in a sequence or proposition. In APNNs it is implemented by a (cyclic) shift [31] or other invertible permutation of item codevector. Similar encoding is considered in [49] and [12].

For our example, it can be represented as **spot**>>shift_agent, **jane**>>shift_object. Here **Z**>>q is the shift of **Z** by q bits, shift_agent and shift_object are the numbers of one-bit shifts corresponding to the agent and object roles.

This type of order representation may be related to symbol-argument-argument (or predicate-arguments) representation (e.g. [17]).

3.2.2. Binding together a set of items

As exemplified in section 3.1, in the HRR scheme a proposition is represented by superimposed codevectors of component sub-propositions and their bindings. In its turn, each sub-proposition is superposition of its own components. It means that in HRRs propositions of all composition levels are represented by a superposition of codevectors which are not bound together.

Unbound representations of propositions, when superimposed, can be confused. For example, the propositions "Spot bit Jane, Fido bit Fred" and "Spot bit Fred, Fido bit Jane" will produce indistinguishable codevectors:

P(bite,spot-jane) + P(bite,fido-fred) = P(bite,spot-fred) + P(bite,fido-jane). (15)

This can be seen from the proper superposition of their **P_structure**s:

P_structure(bite,spot-jane) = bite_agt∗spot + bite_obj∗jane,
P_structure(bite,fido-fred) = bite_agt∗fido + bite_obj∗fred,
P_structure(bite,spot-fred) = bite_agt∗spot + bite_obj∗fred,
P_structure(bite,fido-jane) = bite_agt∗fido + bite_obj∗jane. (16)

The same is true for **P_components**.

It is a kind of superposition catastrophe mentioned in section 2.3.1. Such a scheme limits the flexibility of structure representation, compelling to search a role in the higher-level proposition for binding with each unbound proposition of the lower level. This also prevents from using distributed memory to store propositions (see section 4).

In distinction to HRRs, in APNNs superpositions of codevectors are typically subjected to binding. In particular, all codevectors superimposed in a proposition can be bound together by the auto-CDT procedure. For example, the propositions of Eq. 16 will look as follows:

P_apnn-structure(bite,spot-jane) = [{**bite_agt∨spot**} ∨ {**bite_obj∨jane**}],
P_apnn-structure(bite,fido-fred) = [{**bite_agt∨fido**} ∨ {**bite_obj∨fred**}],
P_apnn-structure(bite,spot-fred) = [{**bite_agt∨spot**} ∨ {**bite_obj∨fred**}],
P_apnn-structure(bite,fido-jane) = [{**bite_agt∨fido**} ∨ {**bite_obj∨jane**}]. (17)

Then the codevectors of propositions "Spot bit Jane, Fido bit Fred" and "Spot bit Fred, Fido bit Jane" will no longer be the same

/P_apnn(bite,spot-jane) ∨ P_apnn(bite,fido-fred)/ ≠
/P_apnn(bite,spot-fred) + P_apnn(bite,fido-jane)/, (18)

because the "thinning brackets" { }, [], // can not be opened.

Let us note that if each proposition Eq. 16 is normalized, the equality in Eq. 15. will not be exact. However, normalization in HRRs has not been considered as binding operation, whereas the CDT procedure binds and normalizes at the same time.

3.2.3. Examples of relational structure representation

Let us consider the peculiarities of possible APNN representations of relational structures from section 3.1.

Using role-filler bindings for order representation, **P_structure(bite)** of Eq. 9 can be represented as in Eq. 17:

P_apnn-structure(bite,spot-jane) = [{**bite_agt**∨**spot**} ∨ {**bite_obj**∨**jane**}]. (19)

However, this representation need not be augmented with **P_components(bite)** as in Eq. 10. Since the CDT procedure preserves unstructured similarity, [{**bite_agt**∨**spot**} ∨ {**bite_obj**∨**jane**}] is already similar to its constituents, and it is also possible to decode the bindings using unstructured similarity. In the same manner, representation of **P(flee)** and **P** can be simplified using APNN representations with role-filler bindings:

P_apnn(bite) = [{**bite_agt** ∨ **spot**} ∨ {**bite_obj** ∨ **jane**}], (20)

P_apnn(flee) = [{**flee_agt** ∨ **jane**} ∨ {**flee_obj** ∨ **spot**}], (21)

P_apnn = **</cause_antc** ∨ **P_apnn(bite)/** ∨ **/cause_cnsq** ∨ **P_apnn(flee)/>**. (22)

Using shift (or other invertible permutation) of codevectors to encode the order, representation can be as follows:

P_apnn1-structure(spot-jane) = **spot**>>agent ∨ **jane**>>object. (23)

However since such a representation is not similar at all to

P_apnn1-structure(jane-spot) = **jane**>>agent ∨ **spot**>>object, it should be augmented with the component representation, in the same manner as for HRRs:

P_apnn1-components(bite,spot,jane) = **bite** ∨ **spot** ∨ **jane**. (24)

So the proposition *bite(spot,jane)* will look as:

P_apnn1(bite) =

{**P_apnn1-structure(spot-jane)** ∨ **P_apnn1-components(bite,spot,jane)**}. (25)

P_apnn1(flee) is constructed in the same manner:

P_apnn1(flee) =

{**P_apnn1-structure(jane-spot)** ∨ **P_apnn1-components(flee,jane,spot)**}, (26)

and the whole proposition looks as:

P_apnn1 = [**cause** ∨ **P_apnn1(bite)** ∨ **P_apnn1(flee)**

∨ **P_apnn1(bite)**>>cause_antc ∨ **P_apnn1(flee)**>>cause_cnsq]. (27)

4. Memory

4.1. Memory organization

Codevectors of items of various composition levels should be stored in memory. It allows to find the closest match for a codevector. It also allows to find the components of a composite item.

In HRRs and in BSCs, fillers of all nesting levels are considered "chunks" and should be stored in memory. For all nesting levels (except may be the lowest one) those chunks or fillers are superpositions of codevectors and bindings of previous level(s).

Codevectors of chunks and sub-chunks of all composition or nesting levels are memorized in one memory array (large-scale clean-up memory [42]). They are used in the

reconstruction of components of complex structures. For example, to decode **P** (Eq. 13), first its component codevector **cause** (or **cause_antc** and **cause_cnsq**) should be retrieved using unstructured similarity. Then, to decode e.g. the cause antecedent, **P** should be unbound using $\mathbf{cause_antc}^T$. Resulting noisy version of **P(bite)** can be cleaned up by accessing the memory and retrieving the closest match. In its turn, reconstructed **P(bite)** can be decoded through the lower level components (as in Eq. 11-12).

In APNNs, the memory is organized hierarchically [29], [33]. Codevectors corresponding to composite items of different composition levels are memorized in different memory arrays. Accordingly, codevectors of the same composition level are memorized in the same memory array.

For the role-filler representation of Eq. 20-22, **spot**, **jane**, **bite_agt**, **bite_obj**, **flee_agt**, **flee_obj**, **cause_antc**, **cause_cnsq** (composition level#1) are stored in memory level#1 (Table 1). Bindings {**spot** $\vee$ **bite_agt**}, {**jane** $\vee$ **bite_obj**}, {**jane** $\vee$ **flee_agt**}, {**spot** $\vee$ **flee_obj**} are stored in memory level#2. Codevectors **P_apnn(bite)** of Eq. 20 (denoted simply as bite in Table 1), **P_apnn(flee)** of Eq. 21 (denoted as **flee**), are stored in memory level#3. Codevectors **/cause_antc** $\vee$ **bite/** = **antc**, **/cause_cnsq** $\vee$ **flee/** = **cnsq** are stored in memory level#4. Codevector **P =Probe** is stored in memory level#5.

To find a similar codevector in memory, a "probe" codevector **P** should be (at least primarily) compared with other available codevectors of its memory level. For example, **P** should be primarily compared with other codevectors of level#5 (so far we have not constructed such codevectors).

To retrieve the components of a composite item, its codevector should be compared with the codevectors of the immediately lower memory level (in other versions, with several lower levels). For example, **P** should be compared with the codevectors of level#4 to find its component bindings **/cause_antc** $\vee$ **P_bite/**, **/cause_cnsq** $\vee$ **P_flee/**. In their turn, they could be decoded through the lower level components, etc. When we reach the codevector {**spot** $\vee$ **bite_agt**} of memory level#2, it should be compared with the codevectors of memory level#1 to find its component codevectors **spot** and **bite_agt**. For decoding, it is not needed to compare a codevector of memory level#2 with codevectors of higher memory levels.

Possible hierarchical levels for the representation of Eq. 23-27 are shown in Table 2.

4.2. Memory implementation

Since in HRRs the chunks are superpositions of codevectors not bound together, the memory where they are stored should not be distributed in order to avoid the risk of superposition catastrophe. Since the memory should return the closest codevector to the input, it should be a memory with error-correcting ability.

Therefore, HRRs require auto-associative memory with non-superposition learning rule. E.g., Plate [42] considers as appropriate the associative memory schemes of Baum, Moody, and Wilczek [3]. In the simplest implementation of "grandmother cell" memory, it contains a number of hidden units. Each unit is devoted to single stored codevector. The input and output weights of a hidden unit are the reference codevector that the unit stores. During retrieval, the hidden unit most close to the noisy input codevector becomes active and outputs the cleaned version of that codevector.

The choice of binary sparse codevectors for APNNs was mainly influenced by the consideration of distributed memory that should be used for their storage and retrieval. It is well-known that distributed memories of matrix type offer a high information capacity for sparse codevectors (e.g. [59], [37], [34], [1], [11], [54]). Since in APNNs the codevectors of

all chunks are thinned, it means they are normalized and bound (see sections 2.3 and 2.5). Normalization implies that the higher level chunks also preserve sparseness and therefore potential for high information capacity. Boundness signifies that the simplest (generic and biologically relevant) superposition learning rules, such as versions of Hebb's learning rule [19], are allowed.

Such a matrix-type auto-associative distributed memory affords good error-correcting ability. Though the maximal capacity is reached at M~logN, we used larger M to support its statistical stability. Even for $N^{1/3} < M < N^{1/2}$, the number of stored codevectors that can be reconstructed from their noisy versions can significantly exceed N (e.g. [44], [45]).

Besides, computational implementation of storage and retrieval operations (vector outer product or vector-matrix multiplication) becomes very simple and fast for binary codevectors, because it does not require at all floating point operations and even multiplication. Further acceleration is possible for sparse codevectors (e.g. [38], [33]). This is all the more true for the operations of finding codevector overlap, superposition, and binding, that are basically reduce to bit operations on codevectors.

These considerations are also true for Kanerva's Sparse Distributed Memory, that is also a candidate for long-term memory implementation [25], [50], [42].

Thus each of the memory arrays mentioned in section 4.1 could be an auto-associative memory of the type discussed in this section. The issues concerning pros and cons of their possible implementation in the same physical memory should be further investigated. However at the current stage of modeling, when not too many items are used, functioning of a memory level can be simulated by storing codevectors of that level in a list and performing an exhaustive comparison for the closest match. In particular, we use such a technique in the experiments of the following sections.

5. Modeling of analogical reasoning

Analogical thinking is one of the most commonly encountered cognitive processes associated with manipulation of structured information. That is why a lot of work is devoted to its study and modeling. (For an introduction, see e.g. [14], [15], [53], [24] and references therein).

Analogs are matched not only by "surface" or "superficial similarity" (see e.g. [13], [9]) based on common analog's elements (attributes, objects, relations, propositions) or by a more broad "semantic similarity" (see e.g. [53], [6]) of those elements. Of no less importance is structural similarity which is determined by how the elements of analogs are arranged with each other. It is based on the notion of "structural consistency" [14], [7] or "isomorphism" [53], [24]. These psychological constraints on structural similarity require one-to-one correspondence (at most one element in one analog corresponds to any element in the other) and parallel connectivity (corresponding relations must have corresponding arguments).

Two first stages in analogical processing are access and mapping. Access (retrieval, recall, recognition) is the process of finding in memory most appropriate source (base) analog(s) given a target (probe, cue) situation. Mapping in the process of finding correspondences between the elements of two analogs.

5.1. Analogical retrieval

Though there are different views on the relative role of surface and structural similarity in analogical retrieval, the models of access which take into account only surface similarity are considered inadequate.

5.1.1. An example of the retrieval task

To illustrate analogical access by estimating similarity of distributed representations, the episodes adapted by Plate [42] from [53] will be used.

The following characters or entities are involved: dogs (Fido, Spot, Rover), people (Jane, John, Fred), a cat (Felix), a mouse (Mort). Relations are bite, flee, cause. The “probe” episode P which is compared to the others is the one used previously as an example for structure representation: "Spot bit Jane, causing Jane to flee from Spot". Other episodes have the same relations as the probe but different types of similarity - mainly according to Gentner's classification ([13], see also e.g. [9]).

Literal similarity (LS) episode has both structural and superficial similarity with the probe: "Fido bit John, causing John to flee from Fido". Surface features (SF) episode has superficial but not structural similarity: "John fled from Fido, causing Fido to bite John". Cross-mapped analogy (CM) episode has both structural and superficial similarity, but types of corresponding entities are switched: "Fred bit Rover, causing Rover to flee from Fred". Analogy (AN) episode has structural but not superficial similarity: "Mort bit Felix, causing Felix to flee from Mort". First order relations only (FOR) episode has neither structural nor superficial similarity, other than shared predicates: "Mort fled from Felix, causing Felix to bite Mort".

Human experimental data converge at that surface similarity is more important for analogical retrieval than structural similarity (experimental results are reported e.g. in [9], [58], [47]). In terms of analogical similarity types it means that analogs of similarity type LS, CM, SF are easier to retrieve than those of similarity type AN and FOR. Among the group of situations with surface similarity, LS is easier accessible than CM and SF. The difference between CM and SF is not clear.

Analyzing data of various authors, Plate [42] provides the following pattern for retrievability of analogs from long-term memory:

$$LS > CM \geq SF > AN \geq FOR. \quad (28)$$

5.1.2. Estimating analogical similarity with HRRs

The scheme for episode encoding was considered in section 3.1. To build HRRs according to that scheme, it is necessary to encode the entities and relations using base-level codevectors. By base-level codevectors we mean here independent codevectors of the lowest composition level (level#0). As mentioned in section 2.1, in HRRs the codevectors of independent items are N dimensional real-valued vectors with the elements chosen randomly and independently using the Gaussian distribution with 0 mean and variance 1/N. After generation, they are fixed. For the experiments reported in [42], Plate used codevectors of N=2048.

All relations and roles were not considered similar, and were represented by the base-level codevectors, e.g. **bite**, **bite_agt**, **bite_obj**, etc. The codevectors of entities (or tokens of some type) were formed as superposition of base-level codevectors of two features: that of type (**human**, **dog**, **cat**, **mouse**) and that of identifier or name (**id_john**, **id_fido**, etc.). For example, the entities John and Mort were represented correspondingly as **john** = **<human+id_john>**, **mort** = **<mouse+id_mort>**.

The similarity scores obtained by Plate [42] are shown in the first column of Table 3. These results show that the similarity of HRRs encoding the episodes is influenced both by their superficial and structural similarity. HRRs are similar is they include similar relations or entities (episodes AN, FOR). The similarity is higher if they include both similar relations and entities (episodes CM, SF). The similarity is still higher if similar entities fill similar roles in

similar relations (episode LS). These results correspond to the experimental data expressed by Eq. (28).

5.1.3. Estimating analogical similarity with APNNs

We have conducted experiments in analogical similarity estimation using APNN representations. Random binary vectors with N=100000, M=1000 were used as the base-level codevectors. Entities were presented as superpositions of base-level codevectors corresponding to the same features as in HRRs of section 5.1.2 (see also level#0 and level#1 of Table 1). Both schemes of section 3.2.3: Eq. 20-22 and Eq. 23-27 were used to construct the representations of episodes.

For the scheme of Eq. 20-22 the propositions of level#2 and higher levels (as in Table 1) were thinned down to 2M. The similarity scores between the codevector of probe and that of each episode are presented in the second column of Table 3.

For the scheme of Eq. 23-27 the proposition of level#2 and level#3 were thinned to 4M. The similarity scores are given in the third column of Table 3.

The order of similarity scores for both types of considered APNN representations is the same as for HRRs and as observed in experiments with people. If it should be necessary to increase the "gap" between the scores, it could be achieved e.g. by decreasing the weight of features-identifiers, that only represent noise in the chosen representation. That may correspond to the transfer to schematic representation.

5.2. Mapping of analogies

The second step in analogy processing after retrieval of analogous episode is usually mapping or interpretation of analogy. When both analogs are known, the task is to find corresponding elements of analogs.

As mentioned in introduction and in section 2.3.1, most of the mapping models (e.g. [7], [22], [21], [24], [6], [16]) use at the final stage a localist mapping network (real or virtual) or its symbolic analog. However, in HRRs and BSCs another technique is used to find corresponding elements of analogs ([39], [42], [27], [28]). This technique is based on operations directly on the distributed representations of analogs.

For example, in order to find the entity in various episodes corresponding to Jane in probe, the following procedure is used [42]. First noisy roles of Jane are extracted from the probe by unbinding: $\mathbf{jane\text{-}roles} = \langle \mathbf{P} * \mathbf{jane}^T \rangle$ + noise.

Then noisy fillers of these roles are extracted from the considered episode **E**, again using unbinding: $\langle \mathbf{E} * \mathbf{jane\text{-}roles}^T \rangle$.

At last, the similarity of those fillers with the episode entities is calculated by dot-product. For the LS episode, the entities are John and Fido, therefore the dot-products $\langle \mathbf{E} * \mathbf{jane\text{-}roles}^T \rangle \bullet \mathbf{john}$, $\langle \mathbf{E} * \mathbf{jane\text{-}roles}^T \rangle \bullet \mathbf{fido}$ are calculated. The entity corresponding to Jane should produce the largest dot-product.

In [42], this procedure produced correct results for the LS and the AN episodes. For the SF, CM, and FOR episodes, the results were incorrect or ambiguous, because in those episodes the similarity of role-filler bindings are inconsistent with the similarity of entities. More sophisticated versions of mapping procedure are needed in order to resolve this ambiguity, e.g. "intermediate clean-up" [42].

5.2.1. Mapping by similarity estimation in APNNs

In order to map the elements of two analogs, first their elements of all levels of hierarchy (or chunks) should be encoded using the APNN representations. Here, let us use the representation scheme of Eq. 20-22.

As proposed in section 4.1, 5 hierarchical levels (plus bas-level) can be distinguished in the of episodes represented by the chosen scheme. The simplest version of mapping we discuss here is to try putting into correspondence the elements of episodes by direct comparison of their codevectors.

Let us consider the similarity matrices calculated for episodes (Tables 4-6). The matrix elements are the overlaps of codevectors of various chunks. It can be seen that for the LS and the AN episodes (Tables 4 and 5) the matrix elements of the main diagonal are the largest. It means that for these episodes the corresponding elements correctly map to each other. Actually, only the similarity between the codevectors inside each level should be taken into account, as indicated by the boxes in the tables. Finally, the entities of level#1 that are the components of the corresponding bindings of level#2 should be also put into correspondence.

For the CM, the SF (Table 6), the FOR episodes, the correspondence can not be established by the similarity of chunks. For those episodes, the similarity of agent/object roles comes into contradiction with the similarity of their fillers. These results of finding correspondences agree with the results of Plate [42] obtained without "intermediate clean-up" and discussed above. It should be noted that such types of analogs are difficult to map even for people without proper experience. To establish "correct" correspondences in the considered cases, probably more elaborated APNN representations or more “formal” techniques for finding correspondences should be used (see also discussion below).

6. Discussion

Estimation of similarity of structures is a key procedure in the models of cognition and in the application systems operating with structured information. Characteristics of this procedure are dictated by the employed form of structure representation.

Earlier, mainly symbolic or local connectionist representations were used for structure representation. Those representations suffer from the problem of information capacity. Each combination of items requires allocation of new memory for its representation as a local unit or a symbol string.

Probably even more pronounced problem is that localist and symbolic representations are semantically brittle or inflexible. For example, every time when some information item is represented, it is necessary to decide should new units be allocated to represent it and its constituents or whether similar existing representations can be exploited. Since local representations are indivisible, it is difficult to represent a gradual degree of similarity or difference between them. A related problem for localist and symbolic representations is the estimation of structure similarity, which requires computationally very expensive procedures.

The problems of structure representation and processing with non-distributed representations are exemplified by the models of analogical reasoning. It should be admitted that in recent models of analogical mapping (LISA [24], STAR2 [16], and especially DRAMA [6]) attempts are made to use semantic flexibility of distributed representations. However, at some stage of implementation these models return to the versions of localist networks that are conceptually similar to the structure handling techniques of earlier prominent models (SME [7], ACME [53], Copycat [21]) based on non-distributed representations.

Computationally expensive stage of structure matching leads to the necessity to introduce two-stage process for retrieval of analogs from memory. Since a lot of potentially analogous situations are stored in memory, it becomes prohibitive to use an expensive process

in comparing the probe with each of them. Therefore a common strategy is to use a computationally cheap process (which does not take structure into account) for selection of the candidates, and then to use a complex computation for estimation of the similarity between the probe and the candidates (taking into account all aspects of similarity), as in MAC/FAC [9] and ARCS [53].

In APNNs, HRRs, and BSCs it has been possible to create an on-the-fly scheme for distributed structure representation, where both the set of components and their arrangements influence the similarity of codevectors. Therefore, similar structures produce similar codevectors. The similarity of codevectors is measured just by the overlap of their 1s (APNNs and BSCs) or dot product (HRRs). Thus it is not needed to search the correspondence between the elements of two structures in order to estimate their overall similarity.

A single-stage process of estimating similarity of structures by the overlap of their APNN-style codevectors was tested on a simple analogical retrieval task. For both schemes of structure representation that were applied, the order of scores for the episodes with different type of similarity corresponded to the order of HRR scores for the same task [42] and to the retrieval pattern observed in people for different similarity types. Connection between the codevector similarity and the probability of human access can be seen as follows: increased similarity between representations of the episode and the probe increases the probability of the episode recall from long-term memory.

The range of possible schemes for structure representations in APNNs is extended by the employed binding procedure and various schemes of order representation. For example, role-filler scheme and predicate-arguments scheme can be used for representation of relational structure. As another example, let us provide possible representation for labeled directed acyclic graph $\phi(\alpha,\psi(\gamma),\psi(\gamma,\phi(\alpha,\beta)))$ [10] (using notation of section 3.2):

$/\phi \vee \alpha>>1 \vee [\psi \vee \gamma>>1]>>2 \vee [\psi \vee \gamma>>1 \vee \{\phi \vee \alpha >> 1 \vee \beta >> 2\}>>2]>>3/.$

Each chunk (composite item, sub-structure, structure) is represented in the APNN scheme by a thinned codevector in which the component codevectors of the lower hierarchical level are bound together. It allows to build the codevector of a higher-level structure simply by superimposing and binding together (by thinning) the component codevectors (without using them as fillers for the roles of still higher-level structure, as in HRRs). It also allows to decrease the risk of confusion when superposition learning rule is used to store the codevectors of composite items.

In APNNs, the hierarchy of representations receives primary attention. The chunk or proposition of each complexity level is memorized and processed in a separate memory array. To retrieve the components of a composite item, the codevectors of the lower hierarchical level are used. Also, the hierarchy allows to simplify the problem of mapping the elements of analogs. The correspondence should be primarily searched between the chunks of the same hierarchical level.

The simplest mapping technique is to put into correspondence the analogical elements of the same hierarchical level having the largest overlap of codevectors. For the encoding of entities and propositions chosen in this paper to represent animal stories, this technique worked successfully for the episodes with literal and analogical similarity. For interpretation of more formal analogies, other representations could be tried: for example, another feature structure of base-level components or different representation scheme for propositional structure.

More structure-sensitive technique for finding correspondences between the elements of analogs is to decode the chunks (starting from the top level) through their sub-chunks of the lower level, to find the correspondences between the roles, and to put into correspondence their fillers - irrespective of their similarity.

Another possible technique is substitution of identical codevectors for the corresponding elements of analogs and re-encoding the analogs. It allows to take into account the information about established correspondences in order to find other correspondences. Also, it can be used to verify the consistency of mapping: if the elements of all levels are mapped correctly and the structure of propositions is identical, then identical codes of the whole structures will result.

Such techniques will be considered elsewhere. They include essentially sequential steps that provide flexibility to the mapping process - though may alternate with parallel stages, e.g. finding most similar codevectors. The process of mapping can be substantially sequential, because it probably works only on a single pair of analogs simultaneously. In distinction, the process of analogical access should allow essentially parallel implementation, because it requires fast retrieval from potentially very huge base of analogical episodes.

In this paper, as in [42], only the simplest flat feature structures (of two features) were used for entities and for relational agents/objects (of a single feature). Such descriptions were generated artificially. Actually, the items of the lowest level may have a complex structure formed in the interaction with environment. For example, the features of *type* are categories that themselves possess a complex hierarchical structure [35]. Therefore the transition to more realistic problems will require to solve the problems concerning meaning representation and symbol grounding ([24], [29], [42], see also [5], [2], [18], [4]).

Further work should reveal how potentially attractive from the scaling point of view schemes for distributed structure representation, processing algorithms exploiting them, and implementations of long-term memory will actually behave themselves for very large numbers of structured items. The APNN scheme seems more neurologically relevant compared to other schemes since it uses distributed sparse binary codes. Besides, manipulations with such codes are very simple - and memory for their storage becomes cheaper.

7. Conclusion

Representation of structures with sparse binary codevectors in the frames of hierarchical architecture of Associative-Projective Neural Networks permits to estimate their aggregate similarity by the overlap of their codevectors. Such a representation has neurological plausibility and allows simple and fast implementation.

The main procedure for codevector construction is that of Context-Dependent Thinning. This procedure binds the codevectors of items together and maintains low density of codevectors of various complexity. This allows to consider distributed auto-associative memory with superposition learning rule as a storage medium for the codevectors of sub-structures of various hierarchical levels.

This “thinning” procedure, as well as some others, also allows to represent the order of arguments in a structure. It offers a potential for representation of various types of data structures, including role-filler and predicate-arguments schemes for representation of relational structure.

The similarity of codevectors representing analogous episodes allows to simulate analogical retrieval from memory. The similarity of codevectors can be also used for analogical mapping, however “difficult” cases may require more complex techniques including similarity-based expansion of higher-level codevectors through their component codevectors.

The concepts and techniques demonstrated in this paper should facilitate the application of distributed representations to the problem of analogy as well as to other structure-sensitive problems of AI that were assumed to be solvable only with symbolic or localist representations.

Acknowledgements: The author would like to thank Ernst Kussul for insightful discussions and comments. This work was supported in part by the International Science Foundation Grants U4M000 and U4M200, as well as by the INTAS Project 93-0560.

References

[1] Amari, S. (1989). Characteristics of sparsely encoded associative memory. Neural Networks, 2, 445-457.
[2] Barsalou, L.W. (in press). Perceptual symbol systems. Behavioral and Brain Sciences.
[3] Baum, E.B., J. Moody, and F. Wilczek (1988). Internal representations for associative memory. Biological Cybernetics 59, 217-228.
[4] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. A. Harshman. Indexing by Latent Semantic Analysis. Jornal of the American Society for Information Science, 41(6): 391-407, 1990.
[5] G. Dorffner, and E. Prem. Connectionism, Symbol Grounding, and Autonomous Agents. Technical Report TR-93-17, Austrian Research Institute for AI.
[6] Eliasmith, C.& Thagard, P. (unpublished) Integrating Structure and Meaning: A Distributed Model of Analogical Mapping.
[7] Falkenhainer, B., Forbus, K.D., & Gentner, D. (1989). The Structure-Mapping Engine: Algorithm and Examples. Artificial Intelligence, 41, 1-63.
[8] 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.
[9] Forbus, K.D., Gentner, D., & Law, K. (1995). MAC/FAC: A model of similarity-based retrieval. Cognitive Science, 19(2), 141-205.
[10] 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.
[11] Frolov, A. A. (1989). Information properties of bilayer neuron nets with binary plastic synapses. Biophysics, 34, 868-876.
[12] 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).
[13] Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. Cognitive Science, 7, pp 155-170.
[14] Gentner, D. & Markman, A. B. (1995). Analogy-Based Reasoning. In M. A. Arbib (Ed.), Handbook of brain theory and neural networks (pp. 91-93). Cambridge, MA: MIT Press.
[15] Gentner, D., & Markman, A. B. (1997). Structure Mapping in Analogy and Similarity. American Psychologist, 52(1), 45-56.
[16] Gray, B., Halford, G. S., Wilson, W. H., & Phillips, S. (1997, September 26-28). A Neural Net Model for Mapping Hierarchically Structured Analogs. Proceedings of the Fourth conference of the Australasian Cognitive Science Society[14], University of Newcastle.
[17] 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.
[18] Harnad, S. (1990). The Symbol Grounding Problem. Physica D 42:335-346.
[19] Hebb, D. O. (1949). The organization of behavior. New York: Wiley.
[20] Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. Artificial Intelligence, 46, 47-76.
[21] Hofstadter, D. R. and M. Mitchel (1988). Conceptual slippage and mapping: A report of the Copycat project. Proceedings of the Tenth Annual Conference of the Cognitive Science Society. Hillsdale, New Jersey, Erlbaum, 601-607.
[22] Holyoak, K. J., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. Cognitive Science, 13, 295-355.
[23] Hopfield, J. J., Feinstein, D. I., & Palmer, R. G. (1983). "Unlearning" has a stabilizing effect in collective memories. Nature, 304, 158- 159.

[24] Hummel, J. E., & Holyoak K. J. (1997). Distributed representations of structure: A theory of analogical access and mapping. Psychological Review, 104, 427-466.
[25] Kanerva, P. Sparse Distributed Memory. MIT Press, 1988.
[26] 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.
[27] Kanerva, P. 1997: "Fully distributed representation." In Proceedings of the 1997 Real World Computing Symposium (RWC'97, Tokyo) pp. 358-365. Real World Computing Partnership, 1997.
[28] Kanerva, P. 1998: "Dual role of analogy in the design of a cognitive computer." In K. Holyoak, D. Gentner, and B. Kokinov (eds.), Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences (workshop proceedings from Analogy'98), pp. 164-170. Sofia: New Bulgarian University.
[29] Kussul, E. M. (1992) Associative neuron-like structures. Kiev: Naukova Dumka. (In Russian).
[30] 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.
[31] 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).
[32] 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.
[33] 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).
[34] Lansner, A., & Ekeberg, O. (1985). Reliability and speed of recall in an associative network. IEEE Trans. Pattern Analysis and Machine Intelligence, 7, 490-498.
[35] Markman, A.B. (1997). Structural alignment in similarity and its influence on category structure. Cognitive Studies, 4(4), 19-37.
[36] Milner, P. M. (1974). A model for visual shape recognition. Psychological Review, 81, 521-535.
[37] Palm, G. (1980). On associative memory. Biological Cybernetics, 36, 19-31.
[38] Palm, G. & Bonhoeffer, T. (1984). Parallel processing for associative and neuronal networks. Biological Cybernetics, 51, 201-204.
[39] Tony A. Plate (1994), ``Estimating structural similarity by vector dot products of Holographic Reduced Representations'', In J. D. Cowan, G. Tesauro, and J. Alspector, editors, Advances in Neural Information Processing Systems 6 (NIPS*93), pp1109-1116, Morgan Kaufmann, San Mateo, CA, 1994.
[40] Plate, T. A. (1995). Holographic Reduced Representations. IEEE Transactions on Neural Networks, 6, 623-641.
[41] 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.
[42] Plate, T. (1998). Analogy retrieval and processing with distributed represenations. Technical report CS-TR-98-4. Wellington, New Zealand: Victoria University of Wellington, School of Mathematical and Computer Science.
[43] Pollack, J. B. (1990). Recursive distributed representations. Artificial Intelligence, 46, 77-105.
[44] 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).
[45] 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).
[46] Rachkovskij, D. A. & Kussul, E. M. (submitted). Binding and Normalization of Binary Sparse Distributed Representations by Context-Dependent Thinning.
[47] Ross, B. (1989). Distinguishing types of superficial similarities: Different effects on the access and use of earlier problems. Journal of Experimental Psychology: Learning, Memory, and Cognition, 15, 456-468.
[48] 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.
[49] Sjodin, G. (1998). The Sparchunk Code: a method to build higher- level structures in a sparsely encoded SDM. In Proceedings of IJCNN'98 (pp.), IEEE, Piscataway, NJ: IEEE.

[50] Sjodin, 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).
[51] Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artificial Intelligence, 46, 159-216.
[52] Sperduti, A. (1994). Labeling RAAM. Connection Science, 6, 429- 459.
[53] P. Thagard, K.J.Holyoak, G.Nelson, D.Gochfeld Analog Retrieval by Constraint Satisfaction Artificial Intelligence 46(1990) 259-310
[54] Tsodyks, M. V. (1989). Associative memory in neural networks with the Hebbian learning rule. Modern Physics Letters B, 3, 555-560.
[55] Touretzky, D. S. (1990). BoltzCONS: Dynamic symbol structures in a connectionist network. Artificial Intelligence, 46, 5-46.
[56] Vedenov, A. A. (1987). "Spurious memory" in model neural networks. (Preprint IAE-4395/1). Moscow: I. V. Kurchatov Institute of Atomic Energy.
[57] 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.
[58] C.M. Wharton, K.J. Holyoak, P.E.Downing, T.E.Lange, T.D.Wickens, and E.R.Melz Below the Surface: Analogical Similarity and Retrieval Competition in Reminding. Cognitive Psychology 26, 64-101 (1994).
[59] Willshaw, D. J., Buneman, O. P., & Longuet-Higgins, H. C. (1969). Non-holographic associative memory. Nature, 222, 960-962.

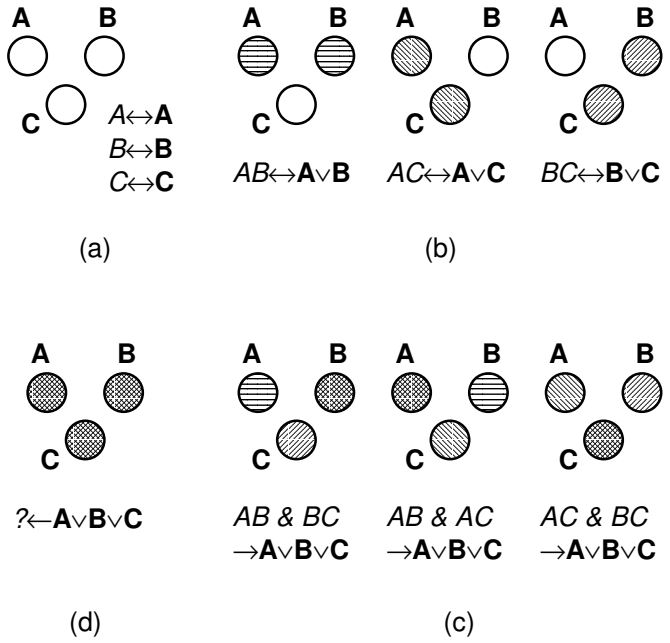

Figure 1. An illustration of "ghosts" or "superposition catastrophe". (a) Each component item *A*,*B*,*C* is represented by the "all-or-none" activity pattern **A**,**B**,**C** correspondingly. (b) Each composite item *AB*, *AC*, *BC* is represented by the superposition of activity patterns of the component items. (c) Superposition of activity patterns of any two of three composite items produces the representation in which the third unforeseen composite item (ghost) is encoded as well. (d) If all three component items are activated, it is impossible to tell which composite items are actually present (superposition catastrophe).

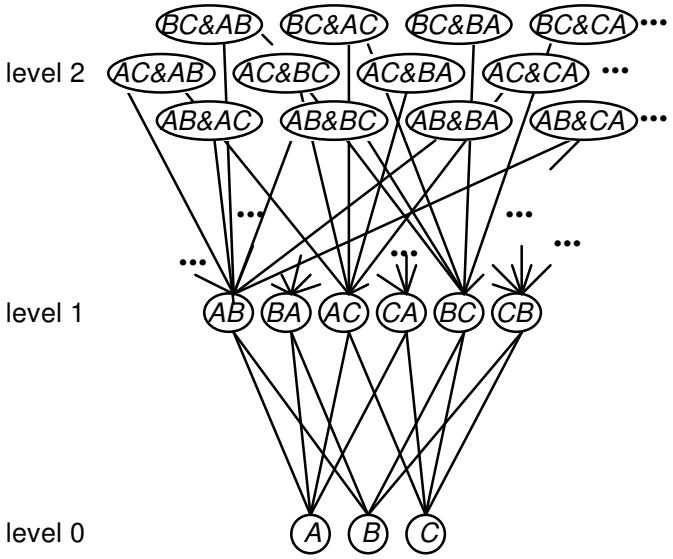

Figure 2. Growth of the number of units which represent possible compositional structures versus the level of compositional hierarchy.

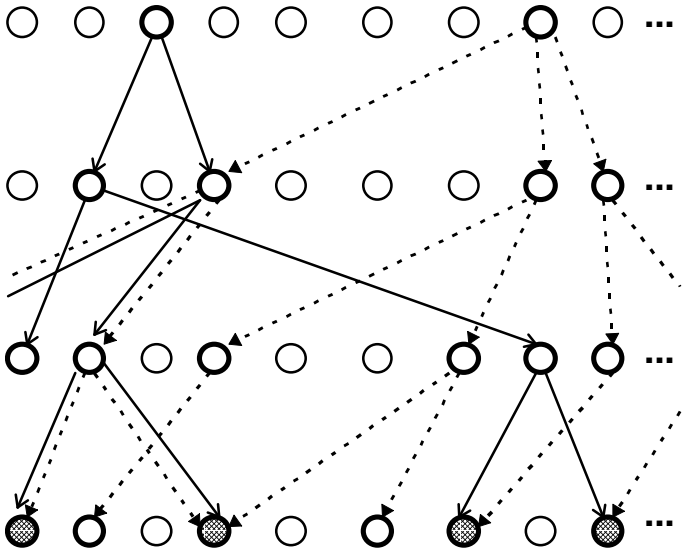

Figure 3. In order to find the mere similarity of the base-level components (even without the structural similarity) of two locally represented hierarchical structures, the nodes representing the whole structures at the top hierarchical level should be unfolded through their component nodes of the lower levels (down to the base-level) using connections between the nodes.

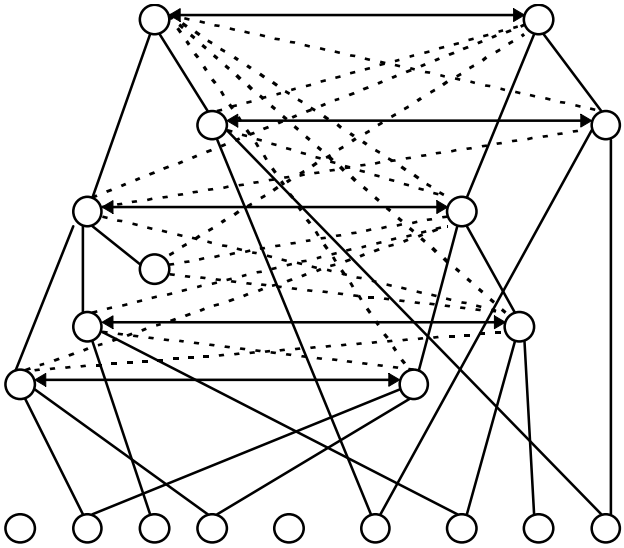

Figure 4. An estimation of the total similarity (including the structural one) of locally represented hierarchical structures requires computationally expensive alignment of their components. The arrows show probably the best match between the component nodes, and the dotted lines show some of possible candidate correspondences.

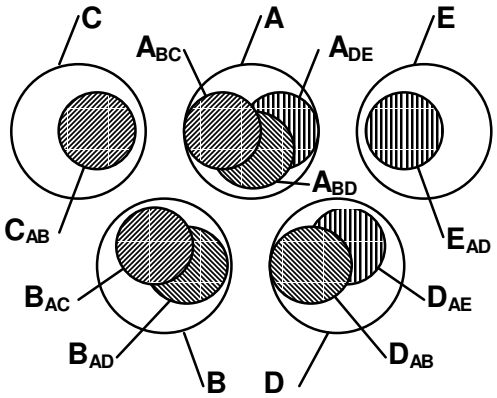

Figure 5. How thinning preserves similarity. Big clear circles represent the 1s encoding the component items *A*, *B*, *C*, *D*, *E*. Representations of three composite items *ABD*, *ABC*, *ADE* formed by thinning are shown by corresponding combinations of three small shaded circles. $\mathbf{X_{YZ}}$ denotes the subset of 1s preserved in the thinned representation of item *X* when items *Y* and *Z* are also present. It can be see that $\mathbf{A_{BC}}$, $\mathbf{A_{BD}}$, and $\mathbf{A_{DE}}$ are all different subsets of **A**, and $\mathbf{A_{BC}}$ is more similar to $\mathbf{A_{BD}}$ than to $\mathbf{A_{DE}}$.

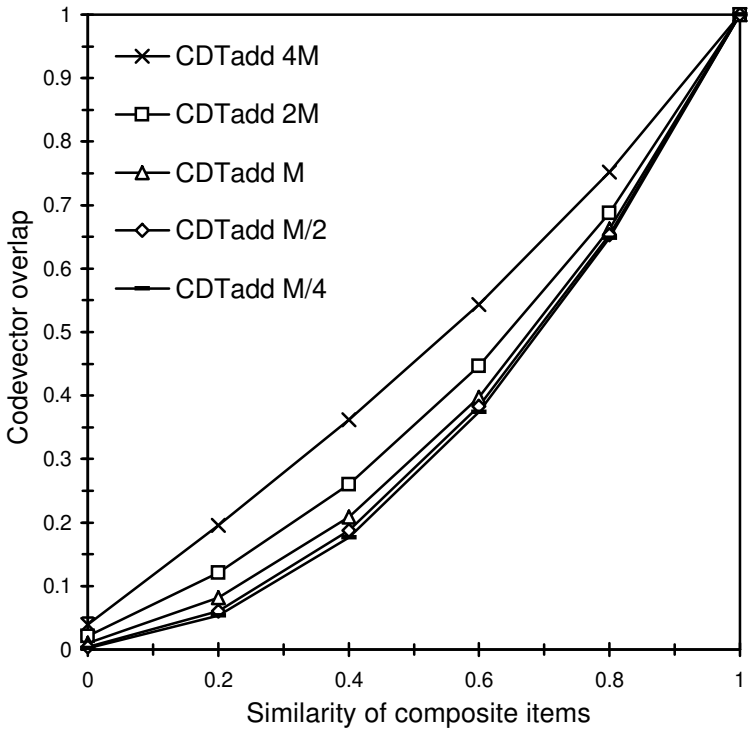

Figure 6. Overlap of thinned codevectors for various "depth" of thinning. There are five components in the composite item. For all component codevectors, N=100000, M≈1000. Therefore the input codevector includes about 4.9M of 1s. The resulting codevector was thinned to have from 4M to M/4 of 1s.

The input codevector $\mathbf{Z} = \mathbf{X}_1 \vee \mathbf{X}_2 \vee ... \vee \mathbf{X}_S$
Set the output (thinned) codevector {**Z**} to 0.
Seed the random numbers' generator: randomize(*seed*).
Note: *seed* defines a unique configuration of shift permutations

```
while(|{Z}| < M)
        if ((r=rand()) ≠ 0)
                for(i=1; i≤N; i=i+1)
                        {Z}_i = {Z}_i ∨ (Z_i ∧ Z_(i+r modulo N));
```

Figure 7. An example of algorithmic implementation of the additive version of the auto-Context-Dependent Thinning procedure

Type of similarity	Episode	Similarity scores HRR [42]	Eq. 20-22	Eq. 23-27
P	Spot bit Jane, causing Jane to flee from Spot	1.00	1.00	1.00
LS	Fido bit John, causing John to flee from Fido	0.71	0.42	0.40
SF	John fled from Fido, causing Fido to bite John	0.47	0.38	0.24
CM	Fred bit Rover, causing Rover to flee from Fred	0.47	0.38	0.30
AN	Mort bit Felix, causing Felix to flee from Mort	0.42	0.26	0.14
FOR	Mort fled from Felix, causing Felix to bite Mort	0.30	0.25	0.09
	Standard deviation range	0.016-0.026	0.009 - 0.013	0.005 - 0.008

Table 2. Similarity scores between the codevectors representing probe and episodes. HRR scores are given in the first column. The scores for two schemes of APNN representation are presented in the second and in the third columns.

Composition or memory level	Codevectors of items or chunks	Configuration of thinning
level#5	**Probe** = < **antc** ∨ **cnsq** >	< >
level#4	**antc** = / **cause_antc** ∨ **bite** / **cnsq** = / **cause_cnsq** ∨ **flee** /	//
level#3	**bite** = [{**bite_agt** ∨ **spot** }∨ {**bite_obj** ∨ **jane**}] **flee** = [{**flee_agt** ∨ **jane**} ∨ {**flee_obj** ∨ **spot**}]	[]
level#2	**bite_a** = {**bite_agt** ∨ **spot**} **bite_o** = {**bite_obj** ∨ **jane**} **flee_a** = {**flee_agt** ∨ **jane**} **flee_o** = {**flee_obj** ∨ **spot**}	{ }
level#1	**spot = dog** ∨ **id_spot** **jane = human** ∨ **id_jane** **bite_agt** **bite_obj** **flee_agt** **flee_obj** **cause_antc** **cause_cnsq**	
level#0 (base-level)	**human** **dog** **cat** **mouse** **id_jane** **id_john** **id_fred** **id_felix** **id_fido** **id_spot** **id_rover** **id_mort** **bite_agt** **bite_obj** **flee_agt** **flee_obj** **cause_antc** **cause_cnsq**	

Table 1. Memory organization for APNN-style representation (Eq. 20-22) of the nested proposition "Spot bit Jane, causing Jane to flee from Spot". The codevector of each level of composition is stored in a memory array corresponding to that level. Codevectors of composite items (level#2 - level#5) are thinned. Non-composite elements of level#1 coincide with the base-level codevectors of level#0.

Composition or memory level	Codevectors of items or chunks	Configuration of thinning
level#3	**P** = [**cause** ∨ **Bite** ∨ **Flee** ∨ **Bite**>>cause_antc ∨ **Flee**>>cause_cnsq]	[]
level#2	**Bite** = {**bite** ∨ **spot** ∨ **jane** ∨ **spot**>>agent ∨ **jane**>>object} **Flee** = {**flee** ∨ **spot** ∨ **jane** **jane**>>agent ∨ **spot**>>object}	{ }
level#1	**spot = dog** ∨ **id_spot** **jane = human** ∨ **id_jane** **bite** **flee** **cause**	
level#0 (base-level)	**human** **dog** **cat** **mouse** **id_jane** **id_john** **id_fred** **id_felix** **id_fido** **id_spot** **id_rover** **id_mort** **bite** **flee** **cause**	

Table 2. Memory organization for APNN-style representation (Eq. 23-27) of the nested proposition "Spot bit Jane, causing Jane to flee from Spot". The codevector of each level of composition is stored in a memory array corresponding to that level. Codevectors of composite items (level#2, level#3) are thinned.

		level#5	level#4		level#3		level#2			
		Probe	**antc**	**cnsq**	**bite**	**flee**	**bite_a**	**bite_o**	**flee_a**	**flee_o**
level#5	**Epis(LS)**	0.41	0.30	0.31	0.20	0.20	0.15	0.15	0.15	0.14
level#4	**antc**	0.30	0.50	0.09	0.29	0.12	0.21	0.21	0.11	0.10
	cnsq	0.30	0.09	0.51	0.12	0.30	0.10	0.11	0.20	0.20
level#3	**bite**	0.19	0.28	0.11	0.41	0.16	0.29	0.29	0.15	0.13
	flee	0.19	0.12	0.28	0.17	0.43	0.14	0.14	0.29	0.29
level#2	**bite_a**	0.14	0.20	0.09	0.28	0.13	0.56	0.02	0.02	0.24
	bite_o	0.14	0.21	0.10	0.28	0.14	0.03	0.55	0.26	0.03
	flee_a	0.15	0.11	0.20	0.14	0.30	0.02	0.26	0.57	0.02
	flee_o	0.15	0.10	0.20	0.13	0.29	0.25	0.02	0.02	0.56

Table 4. The similarity matrix for the elements of the probe and the LS episode

		level#5	level#4		level#3		level#2			
		Probe	**antc**	**cnsq**	**bite**	**flee**	**bite_a**	**bite_o**	**flee_a**	**flee_o**
level#5	**Epis(AN)**	0.25	0.19	0.19	0.08	0.08	0.06	0.07	0.07	0.07
level#4	**antc**	0.19	0.33	0.02	0.11	0.03	0.10	0.10	0.02	0.02
	cnsq	0.20	0.02	0.35	0.02	0.12	0.02	0.03	0.11	0.11
level#3	**bite**	0.07	0.11	0.03	0.16	0.03	0.15	0.14	0.02	0.02
	flee	0.07	0.02	0.11	0.02	0.16	0.02	0.02	0.14	0.14
level#2	**bite_a**	0.06	0.10	0.02	0.15	0.02	0.26	0.02	0.02	0.01
	bite_o	0.06	0.09	0.02	0.13	0.02	0.02	0.25	0.02	0.02
	flee_a	0.06	0.02	0.10	0.02	0.14	0.02	0.02	0.26	0.02
	flee_o	0.07	0.02	0.09	0.02	0.13	0.01	0.03	0.02	0.25

Table 5. The similarity matrix for the elements of the probe and the AN episode

		level#5	level#4		level#3		level#2			
		Probe	**antc**	**cnsq**	**bite**	**flee**	**bite_a**	**bite_o**	**flee_a**	**flee_o**
level#5	**Epis(SF)**	0.38	0.28	0.29	0.19	0.20	0.15	0.14	0.15	0.15
level#4	**antc**	0.28	0.33	0.21	0.10	0.28	0.09	0.09	0.19	0.19
	cnsq	0.27	0.21	0.34	0.28	0.13	0.20	0.20	0.11	0.09
level#3	**flee**	0.19	0.12	0.28	0.16	0.42	0.13	0.14	0.28	0.29
	bite	0.19	0.28	0.11	0.42	0.17	0.28	0.29	0.15	0.13
level#2	**flee_a**	0.14	0.11	0.20	0.14	0.29	0.02	0.25	0.56	0.02
	flee_o	0.15	0.09	0.20	0.13	0.29	0.25	0.02	0.02	0.56
	bite_a	0.14	0.20	0.09	0.28	0.13	0.56	0.02	0.02	0.25
	bite_o	0.14	0.20	0.09	0.28	0.13	0.02	0.56	0.26	0.02

Table 6. The similarity matrix for the elements of the probe and the SF episode