ISSN 1060-992X, Optical Memory and Neural Networks, 2021, Vol. 30, No. 1, pp. 37–50. © Allerton Press, Inc., 2021.

Encoding and Decoding of Recursive Structures in Neural-Symbolic Systems

A. Demidovskij*

Higher School of Economics, Nizhny Novgorod, 603014 Russia

**e-mail: ademidovskij@hse.ru*

Received September 7, 2020; revised December 4, 2020; accepted December 8, 2020

Abstract—One of the ways to join the connectionist approach and the symbolic paradigm is Tensor Product Variable Binding. It was initially devoted to building distributed representation of recursive structures for neural networks to use it as the input. Structures are an essential part of both formal and natural languages and appear in syntactic trees, grammar, semantic interpretation. A human mind smoothly operates with the appearing problems on the neural level, and it is naturally scalable and robust. The question arises of whether it is possible to translate traditional symbolic algorithms to the sub-symbolic level to reuse performance and computational gain of the neural networks for general tasks. However, several aspects of Tensor Product Variable Binding lack attention in public research, especially in building such a neural architecture that performs computations according to the mathematical model without preliminary training. In this paper, those implementation aspects are addressed. A proposed novel design for the decoding network translates a tensor to a corresponding recursive structure with the arbitrary level of nesting. Also, several complex topics about encoding such structures in the distributed representation or tensor are addressed. Both encoding and decoding neural networks are built with the Keras framework's help and are analyzed from the perspective of applied value. The proposed design continues the series of papers dedicated to building a robust bridge between two computational paradigms: connectionist and symbolic.

DOI: 10.3103/S1060992X21010033

1. INTRODUCTION

In the artificial intelligence research community, at least two paradigms were developed parallel for an extended period: symbolic and connectionist. However, the field's future is likely to be connected to coexistence, and even their symbiosis [29].

The traditional symbolic approach is conceived as the continuous development of methods that manipulate symbols. In other words, such methods operate with some explicit representations and aggregate structures that consist of symbols and their combinations. One of the known limitations of this approach is the inability to process data robustly using distributed representation and parallel computations. Simultaneously, symbolic structures are transparent for end-users (for example, fuzzy assessments), there are understandable intermediate results, and, therefore, it is relatively simple to analyze them and perform validation.

The connectionist paradigm is intrinsically built around the concept of massive parallelism [34, 35]. This parallelism is achieved via the usage of multiple uniform units, each performing some simple computational operation and connected selectively to each other. The connectionist level is characterized mostly by artificial neural networks. One of the important aspects of this paradigm is that representations are distributed across computation units, and it is difficult to identify where are particular parts of the input data in that distributed form. This approach is famous for its flexibility and robustness due to its massively parallel nature.

The main challenge is building the bridge between two paradigms to gain benefits from the advantages of both approaches and eliminate, or at least mitigate, cons of each other [2]. The ability of solely connectionist models to perform cognitive tasks and reasoning is substantially debated [29]. Simultaneously, building a fully integrated generic neural-symbolic system is a challenging goal that cannot be achieved with existing approaches and instruments. What is more realistic is building systems capable of performing some particular tasks. For example, one can select the task of multi-criteria linguistic-based decision

making as an appropriate motivating problem [11, 42]. Creation of monolithic neural-symbolic systems for various expert and decision support systems is already a highly demanded and actual task [6, 20, 33].

Information representation is a crucial element of any neural-symbolic system as it defines the way symbolic and connectionist levels communicate with each other. For the past four decades, multiple types of representation were introduced [19]: strictly local, distributed, local, as micro features and by coarse coding. Distributed representations were shown to be more compact and efficient than any type of local representation [27].

This paper considers a special type of distributed representations called Tensor Product Representations [39]. These representations' special property is non-lossy encoding and decoding of recursive symbols, such as binary trees. There is existing mechanism of encoding such symbols in distributed format [9, 14]. This work aims at defining a neural architecture capable of decoding symbols from its distributed Tensor Representation without prior learning. According to [29], it is necessary "to identify a well-motivated set of initial primitives, which might include operations over variables, mechanisms for attention, and so forth". Elaborated neural network complements the neural primitives toolbox of a system architect and, by design, it is a re-usable building block not only for the final decoding of symbols but also for extraction of structural elements of encoded structure.

The structure of the paper is as follows. Section 2 contains brief review of the background study. Then, Section 3 covers the detailed description of the Tensor Product Variable Binding approach to demonstrate its application to the sample recursive structure. After that, Section 4 gives a high level architecture of the Tensor Product Variable neural networks, as well as a proposed design for the decoding network that performs the unbinding task. Finally, Section 5 provides conclusions and defines directions of further research.

2. BACKGROUND STUDY

Building various structures is a natural step during the analysis of both formal and natural languages, which is not bound to the construction of syntactic parse trees but is also required to investigate various phonology aspects, semantic interpretation. In general the considerable question is definition of structural well-formedness [27, 39]. In particular, Smolensky proposed the Harmonic grammar approach that tackles a grammar from the standpoint of specification of building recursive structures on the one hand and a device to construct it on the other. One of the major aspects of the linguistic structures analysis is the validity of the sentence particles' semantic interpretation alongside the sentence structure's usual parsing. Using the Harmonic grammar approach, Smolensky demonstrated the opportunity to identify the Harmony function on both the connectivist and symbolic levels. Maximization of Harmony on the connectivist level allows identifying the sentence's acceptability to the set of soft rules. Apart from the Harmony grammar [27, 39], a similar approach of translating symbolic linguistic structures to the distributed representation with the idea of further manipulation of the structure on the tensor level was proposed as an Active-Passive Net [28]. This neural network demonstrates how the connectivist paradigm is applied to the distributed representation of the sentence and produces the encoded structure containing information about the presence of the active or passive voice in the original sentence as the output. This simulation demonstrates the theoretical capabilities of such solutions.

The task of symbolic structures translation to the distributed representation can be solved with different methods. One of them is using First-Order Logics (FOLs). Formal logics are relatively simple and formally defined languages that use strictly defined rules to express information rigorously. In order to use this method, we need to formulate required knowledge in an appropriate form, and since FOL expressions can be translated to the sub-symbolic level, our knowledge is automatically translated to it and can be used by neural networks [36, 40]. There are plenty of methods that represent different entities as high-dimensional vectors: Holographic Reduced Representations (HRRs), Binary Spater Codes, and so on [4]. Vector Symbolic Architectures (VSA) [18]. It assumes the creation of distributed representations of a fixed size. Therefore, the dimensionality of these hyperdimensional representations [23] hugely depends on the applied task [24, 25], where vectors of different size are needed in order for it to remember information without loss. Elaboration of distributed representations can introduce several requirements. Building distributed representations [26, 30] of the structures that were derived as a result of natural language processing should keep the context for the encoded symbols [3, 5]. Simultaneously, there is a huge need to represent not only atoms, like words but structures consisting of atoms. When applied to more vague scenarios, like Multi-Attribute Linguistic Decision Making, information about the problem situation can be organized in the form of a hierarchy of ontologies that is by definition a symbolic structure and, therefore, can be translated to the sub-symbolic level [41, 43].

Another approach to building sub-symbolic representations is Tensor Product Variable Binding (TPVB) [37]. This method enables building distributed representations of fairly complex symbolic structures, for example, binary trees. The unique property of this approach, compared, for example, to the VSA paradigm, is the non-lossy decoding of structure from such a distributed representation. The second important aspect is that a range of symbolic operations can be effectively expressed as tensor manipulations. The latter attribute allows Tensor Product Variable Binding (TPVB) to become a foundation for neural-symbolic algorithms because Artificial Neural Network can serve as a universal executor of these tensor operations. It is essential to mention that creation of such neural networks that do not require training but allowing to perform complex tasks is a challenging and developing goal [32, 33].

In the framework of a broader research, authors try to elaborate neural-symbolic system for decision support systems and expert systems. It implies the need for such a representation that allows non-lossy encoding and decoding of symbols and the support of symbolic operations as tensor products. Due to the reasons mentioned above, local representations cannot be considered an appropriate choice to represent complex structures. Simultaneously, representations with a fixed representational dimension are not applicable as symbols should be correctly decoded after the neural reasoning. Indeed, by building a neural-symbolic decision-making system, the neural level's result would be an encoded symbolic structure representing the final solution of the original problem. We can compare it with tasks when, for example, decoding is not needed, and distributed representations are used as-is for a classification task when it is a closeness of hyperdimensional vectors that matters [25]. So, it is critical for the binding mechanism to be non-lossy. Therefore, the Tensor Product Representation Binding mechanism was selected as the focus of current research. In the next session, there are details and an example of its application to the sample recursive structure.

3. METHODS: TENSOR PRODUCT VARIABLE BINDING FOR A RECURSIVE STRUCTURE

3.1. Encoding Structure in a Distributed Representation

To understand the key principles used in the proposed design of both encoding and decoding networks, it is important to go through the sample structure's whole computational flow. There is already a study dedicated to elaborating neural design for the simple structure with only one nesting level [9]. The current paper demonstrates how this approach can be scaled for the structure with the arbitrary nesting level.

Tensor Product Variable Binding is a way to transform the symbolic structure into a vector format. According to Paul Smolensky [37] it is built on the simple tensor multiplication operation.

Definition 1. Fillers and roles [37]. Let S be a set of symbolic structures. A role decomposition F/R for S is a pair of sets (F, R), the sets of fillers and roles, respectively and a mapping:

$$\mu_{F/R} : F \times R \mapsto Pred(S); (f, r) \mapsto f/r. \tag{1}$$

For any pair $f \in F, r \in R$, the predicate on S $\mu_{F/R}(f, r) = f/r$ is expressed: f fills role r.

Definition 2. Let s be a symbolic structure that consists of pairs $\{f_i, r_i\}$, where f_i represent a filler and r_i represents a role. Tensor product ψ is calculated in the following way:

$$\psi = \sum_i f_i \otimes r_i. \tag{2}$$

A sample structure that is analyzed in the current paper is shown in Fig. 1. It has a nesting level equal two (Fig. 1). In terms of Tensor Product Variable Binding, this structure has three fillers: A, V, P. Also, the current structure is supposed to represent a binary tree. Therefore there are two distinct basic roles: r_0 that denotes the left child and r_1 that denotes the right child. The root of the tree is denoted as ε. The first step in Tensor Product Variable Binding is defining each filler's particular role in the structure. Given that there are only two basic roles, each filler role is a combination of them. We denote it as a bit string containing either "0" or "1". For example, r_{0101} should be interpreted from left to right as "left child of the right child of the left child of the right child of the root". In the presented structure, filler A plays the role r_0, filler $V - r_{01}$, $P - r_{11}$.

Each filler and role should be mapped with a particular vector representation. There is only one strong requirement: fillers, defined on some vector space V_F, should be linearly independent among each other, and roles defined on some vector space V_R should be independent among each other. Therefore, there is assignment for fillers and roles (3).

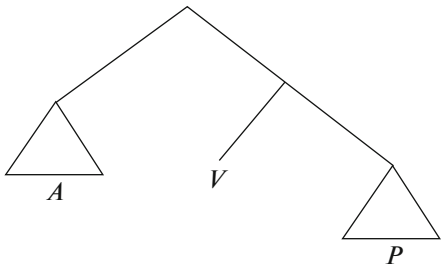


Fig. 1. A recursive structure for demonstration of Tensor Product Variable Binding.

$$\begin{aligned} A &= [7\ 0\ 0\ 0\ 0], \\ V &= [0\ 4\ 0\ 0\ 0], \\ P &= [0\ 0\ 2\ 0\ 0], \\ r_0 &= [10\ 0], \\ r_1 &= [0\ 5]. \end{aligned} \tag{3}$$

Given the notation of roles and fillers, two primitive operations are defined: *cons* and *ex*. The $cons(p,q)$ operation takes two trees as arguments and creates another tree that has tree p as a left child, or in terms of TPRs gets the role (r_0, and q as a right child, or in terms of TPRs gets the role r_1. The vital requirement is to select role vectors so that they are linearly independent. The same requirement applies to the set of fillers vectors as it was proved in [38] the *cons* operation can be expressed as a matrix-vector multiplication.

Definition 3. Let r_0, r_1 denote role vectors, p,q—symbolic structures. Then, joining operation *cons* is defined:

$$cons(p,q) = p \otimes r_0 + q \otimes r_1 = W_{cons_0} p + W_{cons_1} q. \tag{4}$$

Definition 4. Let r_0 denote a role vector, A is a length of any filler vector. Then joining matrix W_{cons_0} is calculated in the following way:

$$W_{cons_0} = I \otimes 1_A \otimes r_0, \tag{5}$$

where I is the identity matrix on the total role vector space, 1_A is the identity matrix $A \times A$. W_{cons_1} matrices are defined in the manner similar to the W_{cons_0} matrices that join two sub-trees in one structure [38]. Extraction operation *ex* is defined analogously. However, it is used to extract an element stored in the tree by the given role. For example $ex_0(p)$ extracts the child of tree p that is placed under role r_0. The only difference in the formulation of W_{ex_0} matrix is that dual role vectors are used instead of direct roles: r_0 or r_1.

Definition 5. Let r_0 denote a role vector, $s = cons(p,q)$—a symbolic structure. Then, extraction operation ex_0 is defined:

$$ex_0(s) = ex_0(cons(p,q)) = p, \tag{6}$$

Definition 6. Let u_0 denote an extraction vector, dual to r_0, A is a length of any filler vector. Then extraction matrix W_{ex_0} is calculated in the following way:

$$W_{ex_0} = I \otimes 1_A \otimes u_0. \tag{7}$$

These matrices are sparse and have a block structure. *cons* operation is used in the encoding procedure and is expressed with a W_{cons} matrices (8), while decoding neural network is built with a weight matrix $W_e x$ (9) representing *ex* operation.

$$W_{cons_0} = \begin{bmatrix} r_{0,0} & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ r_{0,1} & 0 & 0 & \ddots & & & & \vdots \\ 0 & r_{0,0} & 0 & 0 & \ddots & & & \vdots \\ \vdots & r_{0,1} & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & r_{0,0} & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & r_{0,1} & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & r_{0,0} & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & r_{0,1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & r_{0,0} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & r_{0,1} \end{bmatrix}, \tag{8}$$

$$W_{ex_0} = \begin{bmatrix} u_{0,0} & u_{0,1} & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & 0 & u_{0,0} & u_{0,1} & 0 & \ddots & & & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & & & \ddots & \ddots & \vdots \\ \vdots & & & & & & 0 & u_{0,0} & u_{0,1} \end{bmatrix}. \tag{9}$$

Aforementioned operations $cons$, ex_0, ex_1 are equivalent to operations over lists in software general-purpose functional programming languages like Lisp: $cons$, car, cdr. These are universal operators that allow implementation of a considerably large set of algorithms. Considering support of conditional operator, the representational power of these operators is even bigger. Therefore implementation of these operations, or equivalent $cons$, ex_0, ex_1, on the neural level opens the horizon for neural-symbolic computation of symbolic algorithms built on top of $cons$, car and cdr.

According to (2), the tensor representation of the given sample structure should be calculated in the following form (10).

$$\psi = A \otimes r_0 + V \otimes r_{01} + P \otimes r_{11}. \tag{10}$$

After that, we can interpret each complex role as a tensor multiplication of corresponding basic roles (11).

$$\psi = A \otimes r_0 + V \otimes r_0 \otimes r_1 + P \otimes r_1 \otimes r_1. \tag{11}$$

It can be easier to look at the structure from the recursion standpoint and split it into sub-trees until there are only atomic fillers participating in the tensor sum (12).

$$\psi = A \otimes r_0 + (V \otimes r_0 \otimes r_1 + V \otimes r_1 \otimes r_1) = cons(A, cons(V, P)). \tag{12}$$

Regardless the way the tensor representation is formulated (10), (11) or (12), it stated that the Tensor Product Variable Binding should be performed on the sub-symbolic level. In other words, it should happen as a part of a neural network execution. Therefore, the next step would be to prepare compound roles to be used as inputs to the neural network.

The detailed analysis given in [9, 14] shows that it is required to perform pairwise tensor multiplication between pairs of fillers and roles and, finally, find the accumulated sum. The latteris the tensor representation of the given structure S (13).

$$\psi = cons(A, cons(V, P)) = W_{cons_0}A + W_{cons_1}(W_{cons_0}V + W_{cons_1}P) = \begin{bmatrix} 70 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 200 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{13}$$

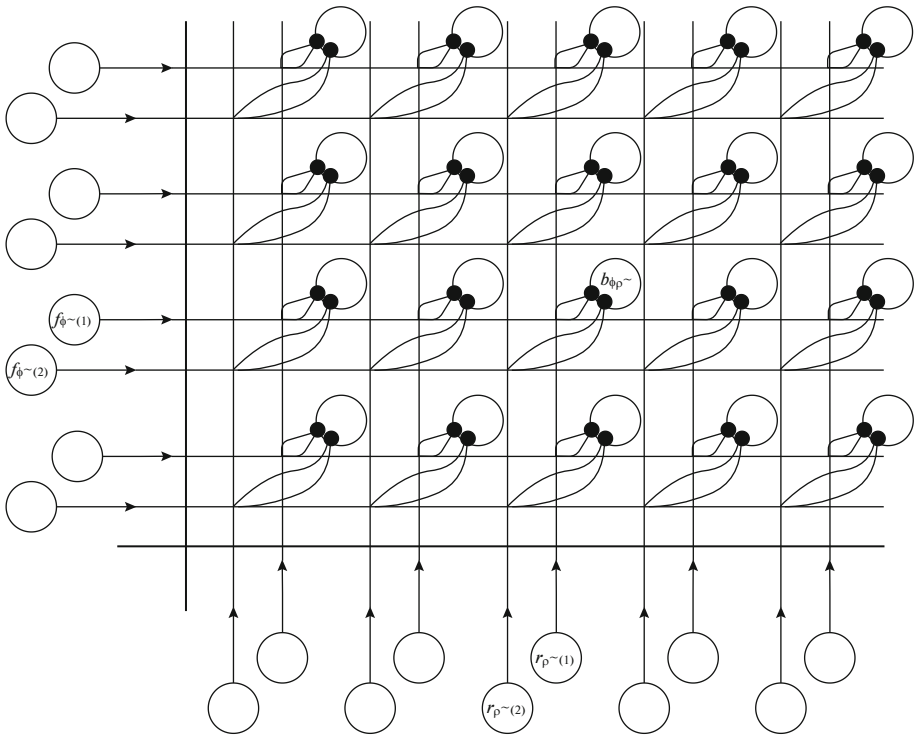


Fig. 2. High-level theoretical architecture for Tensor Product Variable Binding network [37].

3.2. High-Level Theoretical Design of Neural Encoder

The high level design of the neural network that is capable of performing Tensor Product Variable Binding is demonstrated on Fig. 2.

The network is supposed to have two inputs: roles and fillers, and by design, it is a single layer neural network that consists of "sigma-pi" units.

Each sigma-pi unit has several input sites $\{\sigma_i\}$ that are connected with other units in the network. Each site σ_i performs a product of its input connections $\{I_{\sigma i}\}$. An output of the unit is a weighted sum of products from each input site (14). In Tensor Product Variable Binding, weights are not used and are always equal to 1.

$$v = \sum_{\sigma} w_{\sigma} \times \prod_{i} I_{\sigma i} = \sum_{\sigma} 1 \times \prod_{i} I_{\sigma i} = \sum_{\sigma} \prod_{i} I_{\sigma i}. \tag{14}$$

This network is designed to perform the binding operation. It takes fillers and roles and propagates their components through sigma-pi units. The accumulated values at each unit constitute the distributed representation of the given structure. However, the same network can also be used for the unbinding operation. In particular, the network initializes each sigma pi states unit with the corresponding element from the learned tensor representation and accepts vectors of roles as an input. The fillers vector then becomes an output of the network, computed by propagating roles vectors through the existing tensor representation (15).

3.3. Decoding Structure from a Distributed Representation

Imagine there is already a tensor representation of the arbitrary structure. For example, after we have encoded an input structure and performed some computations with it via the neural network, it is expected to decode the structure from the tensor form to evaluate the quality of the result. Therefore, the decoding mechanism is needed.

According to the Tensor Product Variable Binding, it is possible to reveal what filler plays a particular role from the distributed representation. For example, one could consider the task, such as evaluating the

filler positioned as a left-child-of-the-right-child-of-the-root. It is suggested to identify the particular filler from the given role that it plays in the structure.

Definition 7. Unbinding of a filler by its role r_i in the given structure S containing pairs $\{f_i, r_i\}$, which has a tensor representation ψ is calculated in the following way:

$$f_{i,unbinded} = \psi \cdot u_i = \left(\sum_j f_j \otimes r_j\right) \cdot u_i = \sum_j f_j (r_j \cdot u_i) = \sum_j f_j \delta_{ij} = f_i. \tag{15}$$

Vectors u_i are called unbinding vectors for roles r_i. Those unbinding vectors are defined via the basis U_i dual to the basis of primitive roles r_i (17). Each element of this dual basis is a mapping from V_R to either 1 or 0 (16).

$$U_i(r_j) = \delta_{ij} = \begin{cases} 1, & \text{if} \quad i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{16}$$

$$U_i(v) = v \cdot u_i. \tag{17}$$

We assume that there is already a tensor representation and the role that the sought filler plays to solve this task. However, it turns out that in order to find out the filler vector, there is a need to find unbinding vectors for the basic roles vectors. Since the roles matrix is not singular and is square, it is possible to get its inverse (18).

$$\begin{aligned} R &= \begin{bmatrix} 10 & 0 \\ 0 & 5 \end{bmatrix}, \\ U = R^{-1} &= \begin{bmatrix} 0.1 & 0 \\ 0 & 0.2 \end{bmatrix}. \end{aligned} \tag{18}$$

Finally, the unbinding procedure can be made as a matrix-vector multiplication of the extraction matrix W_{ex_0} and the tensor representation of a structure that is flattened as a tensor with rank one (19).

$$A_{decoded} = \psi \times U^T = W_{ex_0} \psi = \begin{bmatrix} 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{19}$$

The decoded filler $A_{decoded}$ fully complies with the filler vectors from the original structure (3) that means that the unbinding procedure was completed without information loss. In the next section, the neural network architecture is covered as well as numerous implementation details.

4. RESULTS AND DISCUSSION

Even though the theoretical organization of neural network (Section 2, Fig. 2) complies with the Tensor Product Variable Binding mathematical model, it is hard for practitioners to express this neural network in terms of modern frameworks and neural networks architecture patterns. This section contains novel architectures for joining and extracting neural primitives.

4.1. Tensor Product Variable Binding Network

Regarding the binding network [9], there were already attempts to design it for a sample structure. Although the very simple structure with a single level of nesting was considered, the network can perform the binding operation over the roles and fillers' pairs.

For the more complex case of recursive structures, the pre-processing step takes the responsibility to translate a structure as a sequence of joining operations (Fig. 3). *cons* operation requires two positional arguments that stand for two structures. However, on the first iterations of encoding, there might be the need to perform the assignment of a new role to a single tree so that there is no counterpart for the joining procedure. In that case, the fake tensor is used, filled with zeros, and has the same rank as the counterpart tree. It is important to note that the tensor's rank that stands for distributed representation of a structure is incremented after each encoding block's execution.

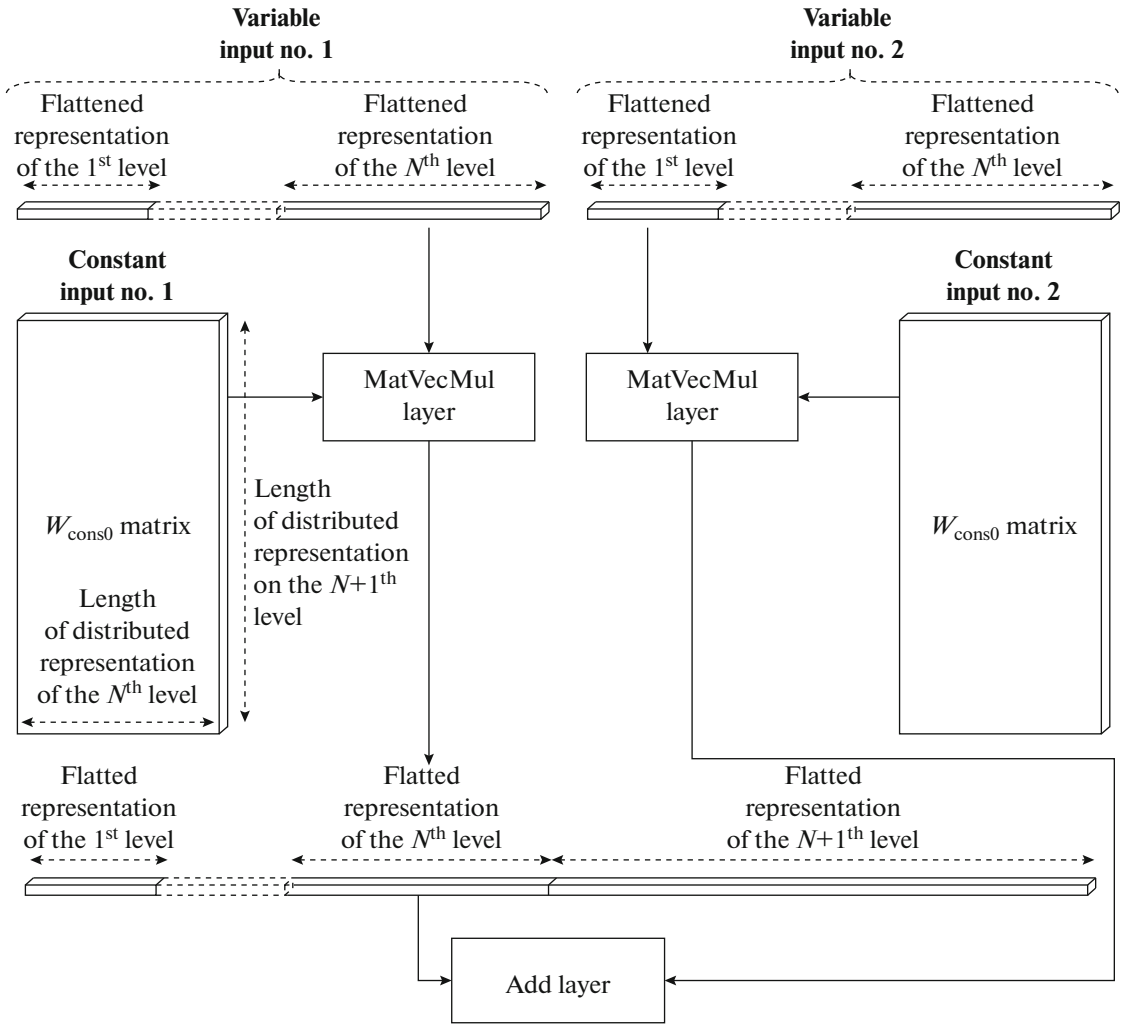


Fig. 3. *cons* operation used for encoding of a binary tree.

4.2. Tensor Product Variable Extraction Network

Another result of the current research is an elaborated design of the unbinding network. It is demonstrated on the Fig. 4. Below main aspects of the topology are considered.

4.2.1. Network inputs. The extraction block accepts two inputs: a variable input[1] that is a distributed representation of a structure with N levels and a constant input that contains a matrix $W_e x$. The latter input is constant as it is created in advance according to the rules defined (9) and the role selected.

4.2.2. Preparing for extraction. Tensor Representations allow encoding of symbolic structures in a distributed representation that is, by definition, a collection of tensors of different rank, where a rank value reflects the nesting level of the structure of a binary tree as it is taken in the example above. Before passing it to the extraction block, flattening variable input is a simple concatenation of vectors flattened versions of these tensors, sorted in ascending order. An important extraction operation procedure is eliminating the vector part that corresponds to the first level of encoded structure. It is a critical part of the decoding block (Fig. 5). When we compare decoder and encoder in terms of design, the former does not require any custom primitive layers and consists only of built-in ones.

4.3. Performance Evaluation

The essential aspect that impacts integration of such neural primitives and Tensor Representations in general is their scalability and the ability to work with truly arbitrary structures, for example, independent

[1] From now on network description contains terminology accepted in the Keras [7] and TensorFlow [1] software frameworks.

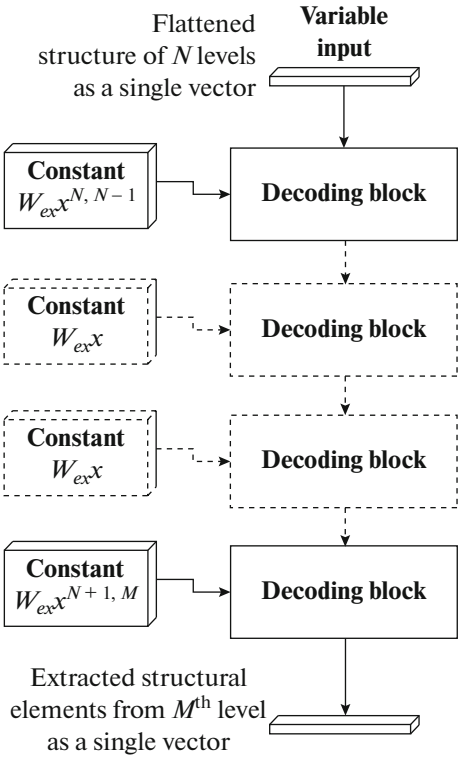


Fig. 4. Proposed design of the decoding network.

of their recursion depth and width if trees are selected as such recursive structures. Overall performance of encoding and decoding tasks is demonstrated in Fig. 6. To benchmark neural encoders and decoders, two additional non-neural Tensor Representations encoding and decoding implementations were made on the foundation of libraries such as NumPy and SciPy. NumPy[2] is a highly optimized library with a broad range of tensor manipulation primitives. SciPy[3] library exposes a rich interface for multiple statistics, machine learning methods, and sparse arithmetics. Therefore, there are three approaches under consideration: Neural, NumPy- and SciPy-based. The neural network was implemented with the Keras framework [7] designed for early prototyping of the neural networks. As an inference engine, the TensorFlow framework was used [1]. Under the hood, the NumPy library was used to construct weight matrices for joining and extraction procedures. Dual roles for extraction procedures are calculated with the help of the SciPy library. Neural encoder and decoder were designed according to the architectures proposed in this paper. The NumPy-based solution is, by definition, a straightforward implementation of Tensor Product encoding and decoding rules that implies the creation of high-order tensors and their products. The SciPy-based solution is fully aligned with the NumPy-based one with the only difference that instead of dense weights matrices creation, we create sparse matrices due to block structure and high sparsity of these tensors by definition.

To reproduce the reported results, the following hardware should be used: Intel®Core™i9-9980HK with frequency 2.40 GHz (not fixed for experiments) and 32 Gb RAM. Libraries should be used with the following versions: Keras—2.2.4, TensorFlow—1.15.2, NumPy—1.19.2, SciPy—1.5.2. Overall implementation is performed using the Python programming language[4]. The source code is available at the open-source repository[5].

The most notable aspect is the inability of neither solution to encode structures of huge depth. More specifically, trees of depth 11 as a maximum can be encoded with Keras implementation. Depth 15 is a limit for NumPy implementation. SciPy is capable of encoding trees of depth 20. The encoding and

[2] http://www.numpy.org/.
[3] https://www.scipy.org/.
[4] https://www.python.org/.
[5] https://github.com/demid5111/ldss-tensor-structures.

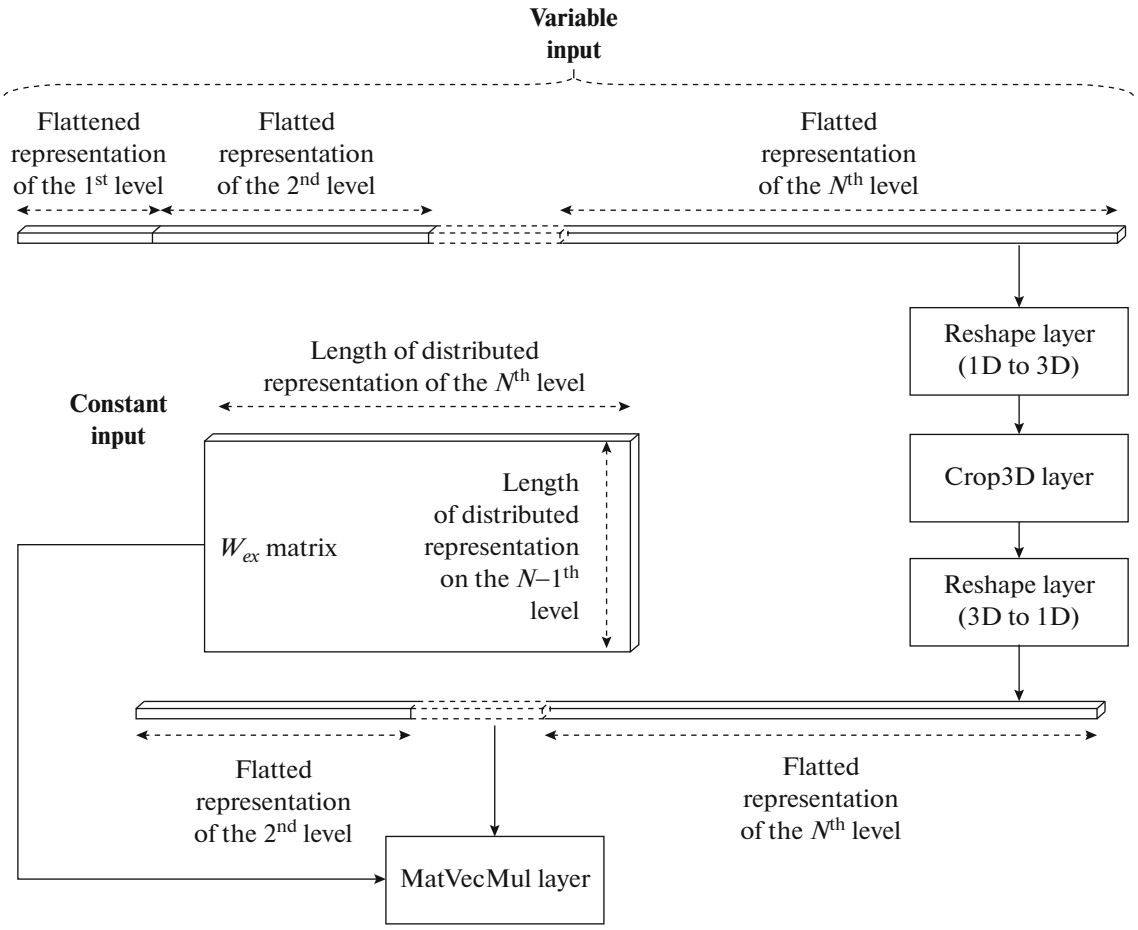
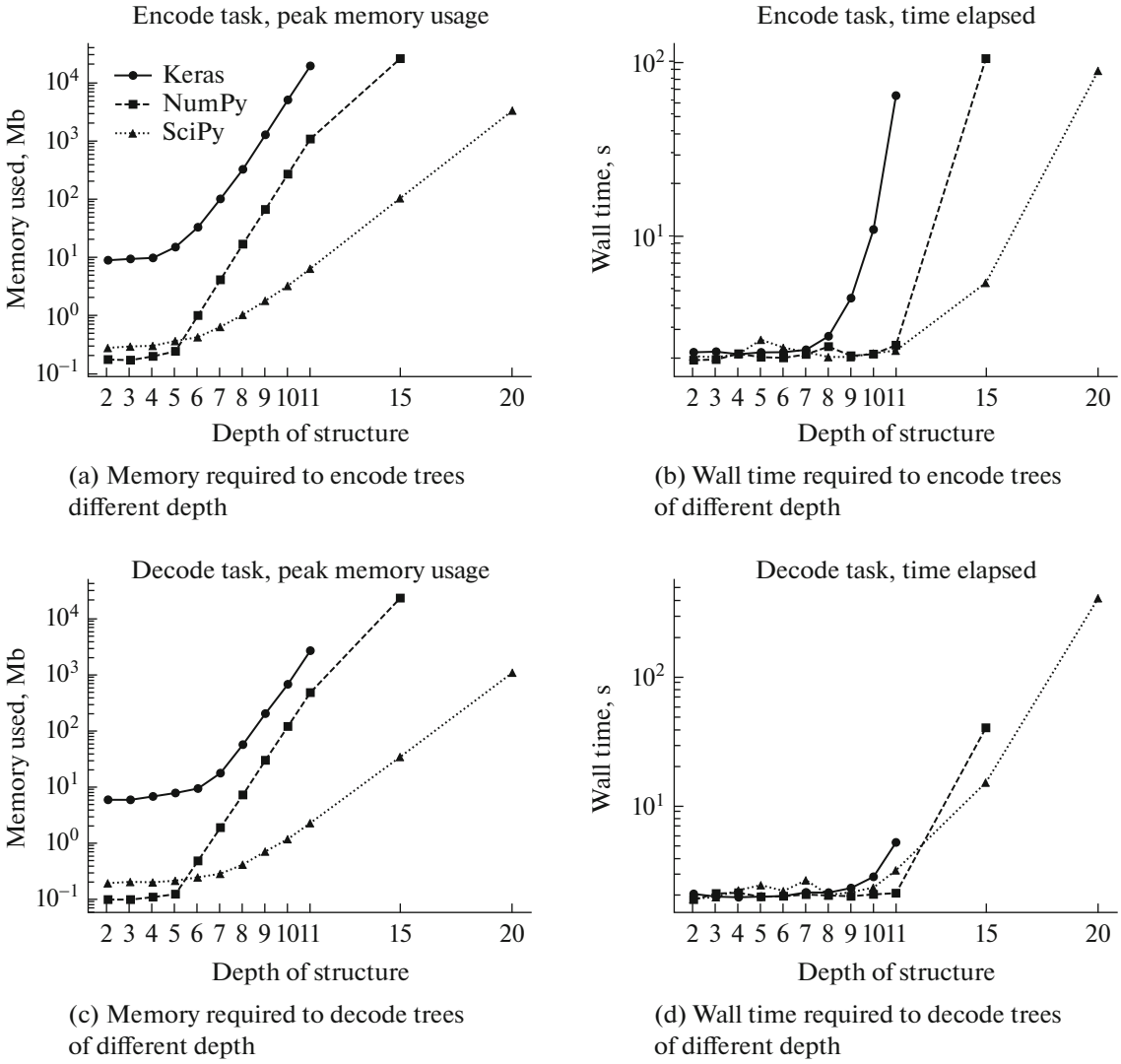


Fig. 5. Decoding block architecture. *ex* operation used for decoding of a structure.

decoding structures' tasks are RAM-bound and require more than 200 Gb disk space for extreme depths of 15 and 20. Simultaneously, encoding and decoding procedures in all implementations are quite consuming in terms of CPU load. Therefore, the first recommendation is to construct symbolic data as a recursive structure with a limited minimum depth while allowing trees to grow in width.

Considering the theoretical availability of unlimited RAM and disk space, there is still a software limit of the libraries, for example, NumPy limit to create arrays with many dimensions more than 32. SciPy also fails to work with huge multi-dimensional arrays. Keras framework does not allow to create tensors with proto larger than 2 Gb. Keras framework does not correctly handle large arrays. Hence, the second recommendation is to design Tensor Representations logic as manipulations of always a one-dimensional array. It requires additional research with proper striding and manipulation instead of a standard outer product available in almost any tensor manipulation library.

The third important observation is the rapid growth of weights matrices sizes as the depth of the encoding structure grows. In particular, to get the left child of a tree root with a depth of 10, the matrix with 5115 rows and 10230 columns is required. At the same time, the matrix is extremely sparse and contains only 5115 elements. Therefore, using sparse matrixes to create and store weight matrixes is beneficial and results in a 10230 times reduction of the required memory to store the matrix mentioned above. Since NumPy does not provide built-in capabilities for working with sparse arithmetics, SciPy was used as an engine for sparse TPR encoding and decoding. Sparsification experiment allows the processing of deeper structures (Fig. 6). Therefore, the recommendation is to build TPR neural encoder and encoder using the latest sparsification advances in the field [22]. Sparsification can be considered a growing trend in the training of neural networks [16, 17]. Building sparse neural networks is also available with the major update of Ten-



(a) Memory required to encode trees different depth

(b) Wall time required to encode trees of different depth

(c) Memory required to decode trees of different depth

(d) Wall time required to decode trees of different depth

Fig. 6. Benchmarking encoding and decoding task across several inference engines: Keras framework with TensorFlow backend, NumPy and SciPy libraries.

sorFlow framework that includes support of sparse matrices[6]. This aspect can be considered as one of the directions of further research and software improvement of the current approach.

5. CONCLUSIONS

To sum up, we have demonstrated the novel design of the extraction network and a generalized joining block that can be re-used easily implemented in any modern framework, for example Keras [7], TensorFlow [1] or PyTorch [31].

These neural encoding and decoding primitives demonstrated in Fig. 3 and Fig. 5 can be massively reused for the creation of neural networks that perform symbolic manipulations. Natural and formal language analysis is tightly coupled with building multiple structures on several levels of abstraction: syntax parsing, semantic interpretation, phonology analysis.

Current research contains proposals on the simplified architecture of the decoding neural network that can perform required operations in parallel and process distributed representations of the arbitrary recursive structures. This neural architecture builds the integrated symbolic and sub-symbolic workflows and can be integrated into broader linguistic frameworks advancing the language theory. When considering

[6] https://www.tensorflow.org/api_docs/python/tf/sparse/SparseTensor.

proposed design as not final networks, but as re-usable blocks, then it opens a broader range of applied tasks such as sentence voice detection [10], neural processing of syntax parse trees [12], neural implementation of integer arithmetics [15], aggregation of linguistic assessments during the decision-making process.

The proposed design of the decoder network keeps all pros of the encoding network:

(1) scalability due to the dynamic number of roles that can be used for unbinding procedure simultaneously;

(2) simplicity due to well-known primitives and clear architectural decisions that can be easily implemented in any modern neural framework

Finally, there is a profound demonstration of both encoding and decoding stages on the example of a recursive structure. An existing neural Tensor Representations encoding proposal [9, 14] lacks such details that are crucial for building robust solution on top of Tensor Product Variable Binding.

The first next step for improving the decoding network's current design is the elimination of unbinding vectors pre-processing. The decoding network expects unbinding vectors to be passed as an input that implies that they are prepared in advance on the pre-processing stage. To make the pure neural computation for the full flow, it is vital to consider elaborating methods of integrating the pre-processing step into the model. It is also crucial to analyze the network performance under load to understand how well this network behaves for the structures with a vast amount of elements and a deep level of nesting.

According to the benchmark results, there are existing software and hardware limitations to encode and decode trees of absolutely arbitrary depth bigger than 20. This obstacle is suggested to overcome by redesigning encoding and decoding rules as operations over one-dimensional arrays with proper striding and manipulation rules instead of implementing Tensor Representations logic based on the mathematical model. Considerable advances in the maximum depth limitation can be achieved using sparse representation of the weight matrices that require less memory and enable the execution of optimized kernels within certain hardware types. It is crucial as the matrices are not randomly sparse but have a clear block structure demonstrated throughout the paper.

However, there is a more fundamental direction for further research. Recently proposed encoding and demonstrated in this paper decoding model do not fully close the gap of building the bridge between connectionist and symbolic levels of computations. Apart from just encoding the network in some distributed representation and then decoding it back to the form of structure, there is a huge need to build such neural networks capable of performing structure manipulations on the tensor level, such as rotating the tree, adding new subtrees. Several papers contribute to this topic regarding building neural-symbolic decision support systems [12, 13]. It was recently proposed to use TPRs for expressing integer arithmetics [15] as a vital part of building those systems. At the same time, there is a promising neural architecture—Neural Turing Machine [21], [21]—that allows training neural networks capable of performing selected arithmetic operations [8]. It seems to authors to be extremely important to compare learned representations versus generated ones using the TPRs rules from the standpoint of capacity.

Bringing both connectionist and symbolic computational paradigms together is an appealing idea, and authors share the belief that such integration is likely to provide significant advances in solving those tasks that can not be solved solely by any of those two approaches, for example, a task of building monolithic neural-symbolic decision support systems.

FUNDING

The reported study was funded by Russian Foundation for Basic Research, project no. 19-37-90058.

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. http://tensorflow.org/.
2. Besold, T.R. and Kühnberger, K.U., Towards integrated neural—symbolic systems for human-level ai: Two research programs helping to bridge the gaps, *Biol. Inspired Cognit. Archit.*, 2015, vol. 14, pp. 97–110.

3. Blacoe, W. and Lapata, M., A comparison of vector-based representations for semantic composition, in *Proc. of the 2012 Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning,* Association for Computational Linguistics, 2012, pp. 546–556.
4. Browne, A. and Sun, R., Connectionist inference models, *Neural Networks,* 2001, vol. 14, no. 10, pp. 1331–1355.
5. Cheng, J., Wang, Z., Wen, J.R., Yan, J., and Chen, Z., Contextual text understanding in distributional semantic space, in *Proc. of the 24th ACM Int. on Conf. on Information and Knowledge Management,* ACM, 2015, pp. 133–142.
6. Cheng, P., Zhou, B., Chen, Z., and Tan, J., The topsis method for decision making with 2-tuple linguistic intuitionistic fuzzy sets, in *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC),* IEEE, 2017, pp. 1603–1607.
7. Chollet, F. et al., *Keras,* 2015. https://keras.io.
8. Collier, M. and Beel, J., Implementing neural turing machines, in *Int. Conf. on Artificial Neural Networks,* Springer, 2018, pp. 94–104.
9. Demidovskij, A., Implementation aspects of tensor product variable binding in connectionist systems, in *Proc. of SAI Intelligent Systems Conference,* Springer, 2019, pp. 97–110.
10. Demidovskij, A., Automatic construction of tensor product variable binding neural networks for neural-symbolic intelligent systems, in *Proc. of 2nd Int. Conf. on Electrical, Communication and Computer Engineering,* IEEE, 2020 (not published).
11. Demidovskij, A. and Babkin, E., Developing a distributed linguistic decision making system, *Business Inform.,* 2019, vol. 13, no. 1.
12. Demidovskij, A. and Babkin, E., Designing a neural network primitive for conditional structural transformations, in *Russian Conf. on Artificial Intelligence,* Springer, 2020, pp. 117–133.
13. Demidovskij, A. and Babkin, E., Designing arithmetic neural primitive for sub-symbolic aggregation of linguistic assessments, *J. Phys.: Conf. Ser.,* 2020, vol. 1680.
14. Demidovskij, A.V., Towards automatic manipulation of arbitrary structures in connectivist paradigm with tensor product variable binding, in *Int. Conf. on Neuroinformatics,* Springer, 2019, pp. 375–383.
15. Demidovskij, A.V. and Babkin, E.A., Towards designing linguistic assessments aggregation as a distributed neuroalgorithm, in *2020 XXIII Int. Conf. on Soft Computing and Measurements (SCM),* IEEE, 2020, pp. 161–164.
16. Dettmers, T. and Zettlemoyer, L., Sparse networks from scratch: Faster training without losing performance, arXiv preprint, 2019. arXiv:1907.04840.
17. Evci, U., Gale, T., Menick, J., Castro, P.S., and Elsen, E., Rigging the lottery: Making all tickets winners, in *Int. Conf. on Machine Learning,* PMLR, 2020, pp. 2943–2952.
18. Gallant, S.I. and Okaywe, T.W., Representing objects, relations, and sequences, *Neural Comput.,* 2013, vol. 25, no. 8, pp. 2038–2078.
19. van Gelder, T., Distributed vs. local representation, *PhD Thesis*, Univ. Pittsburgh, 1999.
20. Golmohammadi, D., Neural network application for fuzzy multi-criteria decision making problems, *Int. J. Prod. Econ.,* 2011, vol. 131, no. 2, pp. 490–504.
21. Graves, A., Wayne, G., and Danihelka, I., Neural turing machines, 2014. arXiv:1410.5401.
22. Gray, S., Radford, A., and Kingma, D.P., Gpu kernels for block-sparse weights, 2017. arXiv:1711.09224 3.
23. Kanerva, P., Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors, *Cognit. Comput.,* 2009, vol. 1, no. 2, pp. 139–159.
24. Kleyko, D., Khan, S., Osipov, E., and Yong, S.P., Modality classification of medical images with distributed representations based on cellular automata reservoir computing, in *2017 IEEE 14th Int. Symp. on Biomedical Imaging (ISBI 2017),* IEEE, 2017, pp. 1053–1056.
25. Kleyko, D., Rahimi, A., Rachkovskij, D.A., Osipov, E., and Rabaey, J.M., Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics, *IEEE Trans. Neural Networks Learning Systems,* 2018, vol. 29, no. 12, pp. 5880–5898.
26. Le, Q. and Mikolov, T., Distributed representations of sentences and documents, in *Int. Conf. on Machine Learning,* 2014, pp. 1188–1196.
27. Legendre, G., Miyata, Y., and Smolensky, P., *Harmonic Grammar: A Formal Multi-Level Connectionist Theory of Linguistic Well-Formedness, Theoretical Foundations,* Citeseer, 1990.
28. Legendre, G., Miyata, Y., and Smolensky, P., Distributed recursive structure processing, *Advances in Neural Information Processing Systems 3 (NIPS 1990),* 1991, pp. 591–597.
29. Marcus, G., The next decade in ai: four steps towards robust artificial intelligence, 2020. arXiv:2002.06177.
30. Mikolov, T., Chen, K., Corrado, G., and Dean, J., Eficient estimation of word representations in vector space, 2013. arXiv:1301.3781.
31. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A., *Automatic Differentiation in Pytorch,* 2017.

32. Pinkas, G., Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge, *Artif. Intell.,* 1995, vol. 77, no. 2, pp. 203–247.
33. Pinkas, G., Lima, P., and Cohen, S., Representing, binding, retrieving and unifying relational knowledge using pools of neural binders, *Biol. Inspired Cognit. Archit.,* 2013, vol. 6, pp. 87–95.
34. Rumelhart, D.E., Hinton, G.E., McClelland, J.L., et al., A general framework for parallel distributed processing, *Parallel Distrib. Process.: Explor. Microstruct. Cognit.,* 1986, vol. 1, no. 45–76, p. 26.
35. Rumelhart, D.E., McClelland, J.L., Group, P.R., et al., *Parallel Distributed Processing,* Vol. 1, Cambridge, MA: MIT Press, 1987.
36. Serafini, L. and Garcez, A.d., Logic tensor networks: Deep learning and logical reasoning from data and knowledge, 2016. arXiv:1606.04422.
37. Smolensky, P., Tensor product variable binding and the representation of symbolic structures in connectionist systems, *Artif. Intell.,* 1990, vol. 46, no. 1–2, pp. 159–216.
38. Smolensky, P. and Legendre, G., *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar (Cognitive Architecture),* MIT Press, 2006, vol. 1.
39. Smolensky, P., Legendre, G., and Miyata, Y., Integrating connectionist and symbolic computation for the theory of language, *Curr. Sci.,* 1993, pp. 381–391.
40. Teso, S., Sebastiani, R., and Passerini, A., Structured learning modulo theories, *Artif. Intell.,* 2017, vol. 244, pp. 166–187.
41. Wang, H., Dou, D., and Lowd, D., Ontology-based deep restricted boltzmann machine, in *Int. Conf. on Database and Expert Systems Applications,* Springer, 2016, pp. 431–445.
42. Wei, C. and Liao, H., A multigranularity linguistic group decision-making method based on hesitant 2-tuple sets, *Int. J. Intell. Syst.,* 2016, vol. 31, no. 6, pp. 612–634.
43. Widdows, D. and Cohen, T., Reasoning with vectors: A continuous model for fast robust inference, *Logic J. IGPL,* 2014, vol. 23, no. 2, pp. 141–173.