Towards Automatic Manipulation of Arbitrary Structures in Connectivist Paradigm with Tensor Product Variable Binding

Alexander V. Demidovskij(✉)

Higher School of Economics, ul. Bolshaya Pecherskaya 25/15, Nizhny Novgorod, Russia
ademidovskij@hse.ru

Abstract. Building a bridge between symbolic and connectionist level of computations requires constructing a full pipeline that accepts symbolic structures as an input, translates them to distributed representation, performs manipulations with this representation equivalent to symbolic manipulations and translates it back to the symbolic structure. This work proposes neural architecture that is capable of joining two structures which is an essential part of structure manipulation step in the connectionist pipeline. Verification of the architecture demonstrates scalability of the solution, a set of advice for engineering practitioners was elaborated.

Keywords: Connectionism · Tensor computations · Neural networks · Unsupervised learning

1 Introduction

For a long period, Artificial Intelligence (AI) community investigates two important paradigms about computations: symbolic and sub-symbolic or connectionist approaches. Although, those two ideas can be considered drastically different, it is likely for them to become partners rather than competitors. Symbolic level is defined by methods that manipulate symbols and explicit representations. Connectionist approach [1, 2] is built around the idea of massive parallelism and mostly characterized by artificial neural networks. The potential symbiosis of two paradigms can bring robust and flexible solutions that produce understandable results that are easy to validate.

Symbolic structures can be encoded in the distributed representation with many means: First-Order Logics (FOLs) [3, 4], Holographic Reduced Representations (HRRs), Binary Spater Codes and so on [5]. One of the key contributions to the field are presented in the Tensor Product Variable Binding approach proposed by Smolensky [6] and further applied in Vector Symbolic Architectures (VSA) [7]. Distributed representations taken by this method are used in multiple domains, especially in Natural Language Processing (NLP) [8], where a sentence plays a role of structure. In order to describe the task and the proposed solution it is essential to give several key definitions of the Tensor Product Variable Binding (TPVB).

B. Kryzhanovsky et al. (Eds.): NEUROINFORMATICS 2019, SCI 856, pp. 375–383, 2020.
https://doi.org/10.1007/978-3-030-30425-6_44

Definition 1. Filler – a particular instance of the given structural type.

Definition 2. Role – a function that filler presents in a structure.

Definition 3. Tensor multiplication is an operation over two tensors a with rank x and b with rank y that produces a tensor z has rank $x + y$ and it consists of pair-wise multiplications of all elements from x and y.

Definition 4. Tensor product of a structure. A structure is perceived as a set of pairs of fillers $\{f_i\}$ and roles $\{r_i\}$ and its tensor product is found as (1).

$$\psi = \sum\nolimits_i f_i \otimes r_i \tag{1}$$

There are already solutions that can translate simple structures to tensor representations and back to the symbolic structures [9]. However, there is a gap in making operations over structures on the tensor level. Indeed, there are multiple routine operations over structures: adding or removing nodes, joining structures together etc. In this paper the task of joining structures together is considered and thoroughly analyzed.

2 Task Description

There is structure **S** presented on the Fig. 1. It consists of two levels of nesting (root is not considered as a first level). This structure contains 3 fillers: **A**, **B**, **C** and only two elementary roles: r_0 (left child) and r_1 (right child). Each filler and role should be transformed to vector representation. There is only one strong requirement: fillers, defined on vector space V_F, should be linearly independent among each other, as well as roles, defined on vector space V_R. At the same time, an assignment for fillers and roles can be arbitrary with the aforementioned condition being satisfied (2).

$$A = [8\ 0\ 0], B = [0\ 15\ 0], C = [0\ 0\ 10], r_0 = [10\ 0], r_1 = [0\ 5] \tag{2}$$

According to Definition 4 the given structure S can be translated to the distributed representation (3).

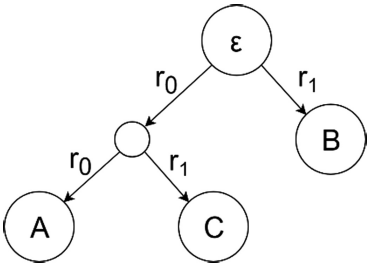

Fig. 1. Sample structure

$$\psi = \sum_i f_i \otimes r_i = A \otimes r_0 \otimes r_0 + C \otimes r_1 \otimes r_0 + B \otimes r_1 \tag{3}$$

It is easier to first calculate compound roles (4) and then apply them to (3) in order to find the corresponding tensor representation (5).

$$\begin{aligned} r_{00} &= r_0 \otimes r_0 = [10\ 0] \otimes [10\ 0] = \begin{bmatrix} [100 & 0] \\ [0 & 0] \end{bmatrix} \\ r_{10} &= r_1 \otimes r_0 = [0\ 5] \otimes [10\ 0] = \begin{bmatrix} [0 & 0] \\ [50 & 0] \end{bmatrix} \end{aligned} \tag{4}$$

$$\begin{aligned} &\psi = A \otimes r_{00} + C \otimes r_{10} + B \otimes r_1 \\ &= [8\ 0\ 0] \otimes \begin{bmatrix} [100 & 0] \\ [0 & 0] \end{bmatrix} + [0\ 0\ 10] \otimes \begin{bmatrix} [0 & 0] \\ [50 & 0] \end{bmatrix} \\ &+ [0\ 15\ 0] \otimes [0\ 5] = \\ &= \begin{bmatrix} \begin{bmatrix} [800 & 0] \\ [0 & 0] \end{bmatrix} & \begin{bmatrix} [0 & 0] \\ [0 & 0] \end{bmatrix} & \begin{bmatrix} [0 & 0] \\ [500 & 0] \end{bmatrix} \end{bmatrix} + \begin{bmatrix} [0\ 0] \\ [0\ 75] \\ [0\ 0] \end{bmatrix} \end{aligned} \tag{5}$$

It is extremely important to note that the resulting tensor representation contains tensors of different rank that cannot be summed as plain matrixes. Instead there is a direct sum operation. The idea is that a tensor of rank **N** can be represented as a list of tensors of rank 1..**N** with tensors of rank 1..**N**−1 being just filled with zeros. Therefore, when a sum of tensor representation is performed tensors are summed according to their rank.

At this moment, it is clear how to build a binary tree of the predefined height using sub-symbolic operations. In order to better understand the requirements of the task of the paper it is necessary to analyze the algorithm that is used to construct the considered example (Fig. 2).

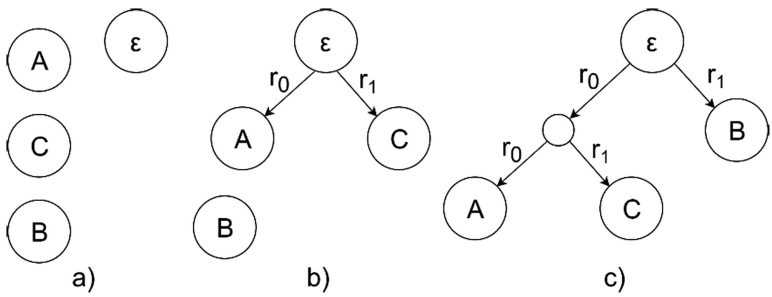

Fig. 2. Possible stages of building structure from subtrees. (a) There are independent fillers. (b) A and C are joined as left and right children of root accordingly. B is still an independent filler. (c) A subtree from (b) is taken as a left subtree and a free filler B is taken as a right subtree.

From Fig. 2 it is clear that building a structure inherently means joining subtrees. In case of binary tree there are one or two subtrees that can be joined. Also, it is vital that at the beginning each filler is considered as a separate tree that can participate in the joining procedure.

This brings to the formulation of the task. The target task of the current paper is to propose the robust neural architecture for performing dynamic construction of tensor representation of the arbitrary structure via joining the subtrees and investigate engineering aspects of its implementation.

3 Theoretical Method of Building Shift Matrix

Joining two subtrees as direct children of the new root and by that constructing the new tree is by nature a simple operation that makes a whole subtree play a new role in terms of Tensor Product Variable Binding. It is extremely clear from Fig. 2b, where instead of taking big trees, there are only two fillers that play a role of left and right subtree correspondingly. In order to achieve the same result on tensor level it is enough to perform tensor multiplication of the filler and corresponding role. Generalizing it to the case when instead of a filler there is a representation of a tree, there is still a need to perform tensor multiplication of the tree distributed representation and the assigned role. The complexity in this case lies in the fact that tensor representation of the structure is the multi-component list of tensors of different depth and it is no longer a plain vector-vector multiplication.

Definition 5. Joining operation cons(p, q) is an action over two structures (trees) so that the tree p is sliding as a whole 'down to the left' so that its root is moved to the left-child-of-the-root position and tree q is sliding 'down to the right'.

Operation *cons* can be expressed for binary trees as:

$$\begin{aligned} cons(p,q) &= p \otimes r_0 + p \otimes r_1 \\ cons_0(p) &\equiv cons(p, \emptyset) \\ cons_1(q) &\equiv cons(\emptyset, q), \end{aligned} \tag{6}$$

where r_0 and r_1 are roles, $\emptyset$ is empty tree.

It was proved [10] that this operation can be expressed in matrix form given that it operates over the tensor representation of structures (7).

$$cons(p,q) = W_{cons0}p + W_{cons1}q \tag{7}$$

Matrix exposes a shifting mechanism over a tensor representation of a structure that contains tensors of different rank. Technically, to shift the tree 'down to the left' ('down to the right') means to apply the role r_0 (r_1) to each tensor from tensor representation. This is what W_{cons0} (W_{cons1}) matrices perform. These matrices take symbols at depth **d** from **p** and put them at depth **d** + 1.The form of these matrices is the following: all elements are zeros except the elements under the main diagonal. This is true because of the fact that *cons* operation just shifts the tree one level down. As both

matrices are constructed in the same manner, only W_{cons0} is considered in this section. Matrix is computed from the role vector and identity matrices (8).

$$W_{cons0} = 1_A \otimes r_0 + 1_R \otimes 1_A \otimes r_0 + 1_R^{\otimes 2} \otimes 1_A \otimes r_0 + \\ + \ldots + 1_R^{\otimes d} \otimes 1_A \otimes r_0 + \ldots, \tag{8}$$

where d is the depth of the representation, 1_A is an identity matrix of width and height equals number of elements in the filler vector and 1_R is an analogous identity matrix with size depending on the role vector.

The key point in constructing the matrix is to keep the order of tensor multiplications. This is not so obvious because the way tensor representation is considered in TPVB is rather unbounded – TPVB only recognizes the feature that resulting tensor contains all multiplications of input tensors elements. However, for W_{cons0} it is very important to keep dimensions of roles first. Finally, we get the following matrix for depth = 2, role vector with 2 elements, filler vector with 3 elements (9).

$$\begin{bmatrix} \begin{bmatrix} \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} & 0 & 0 \\ 0 & \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} & 0 \\ 0 & 0 & \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} \end{bmatrix} & 0 & 0 \\ 0 & \begin{bmatrix} \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} & 0 & 0 \\ 0 & \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} & 0 \\ 0 & 0 & \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} \end{bmatrix} & 0 \\ 0 & 0 & \begin{bmatrix} \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} & 0 & 0 \\ 0 & \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} & 0 \\ 0 & 0 & \begin{bmatrix} r_{0_0} \\ r_{0_1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{9}$$

During the computation phase the matrix is flattened and does not contain the block structure present in (9). Blocks are shown for better visualization of the matrix structure.

4 Proposed Neural Architecture

The overall scheme of the proposed neural architecture for joining structures is demonstrated on the Fig. 3. Neural Network is designed to accept multiple inputs of two types: constant and variable ones, they will be described later. After that each filler

processing subtree is flattened to a vector format while a shifting matrix is prepared based on the role that is chosen for this sub-tree. Finally, each subtree vector representation is multiplied by the shifting matrix and all the resulting vectors are summed and by that the tensor representation of the structure that contains inputs structures as direct children of the new root is produced. All the layers details are covered below.

Input Layers. As it was stated in (4) tensor representation is by definition a list of tensors. Number of elements in the list hugely depends on the depth of the structures that should be joined. Each variable input corresponds to the tensor of the particular rank. Also, there can be multiple structures that we are going to join, that is why the number of inputs can drastically grow with the demand of the original task. The second type of inputs is constant inputs. Those inputs are filled with roles vectors. On the Fig. 2 it is clear that there are only two roles taken for simplicity of description. In reality there can be plenty of roles and Neural Network is designed to be easily extended to a larger case.

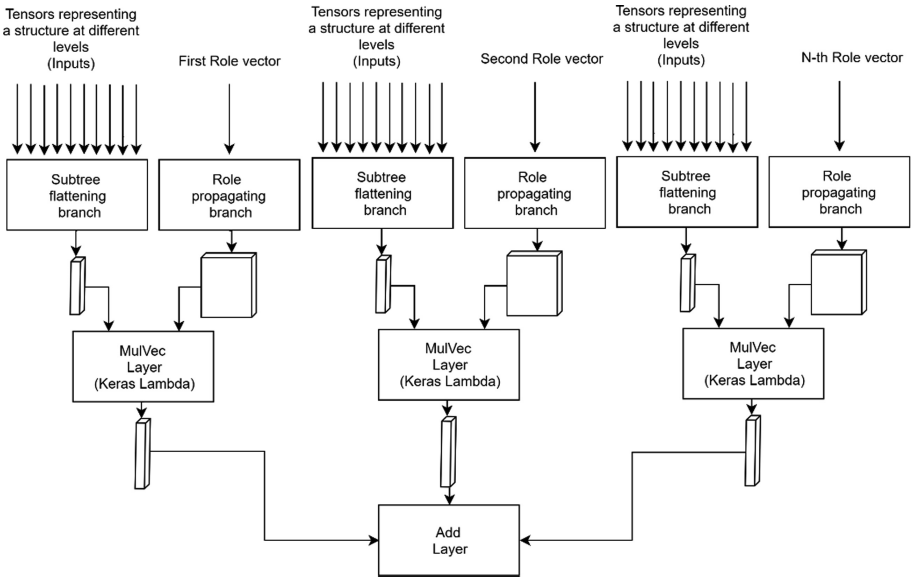

Fig. 3. Overall scheme of the neural architecture

Reshaping Layers. Those layers are part of the subtree flattening branch (Fig. 4) and exist for input tensors or rank 1 and 2. It is a technical requirement of the implementation in the Keras[1] framework due to the fact that Flatten layer can work only with tensors of rank bigger than two. So, Reshaping layers expand dimensions of such inputs with fake dimension of 1 to satisfy Flatten layer requirements.

[1] https://keras.io/.

Flattening Layers. Those layers are part of the subtree flattening branch (Fig. 4) and exist for all input tensors. Those layers transform tensors of different rank to a simple vector format according to the ordinary rules of flattening multi-dimensional tensors.

Concatenate Layers. Those layers are part of the subtree flattening branch (Fig. 4). Those layers join vectors that correspond to each level of the tensor representation in one vector. The order is very important here: from vectors representing zero depth level to **N**.

Transpose Layers. Those layers are part of the subtree flattening branch (Fig. 4). Due to the fact that next operation is matrix-vector multiplication it is required to transform a vector into a column vector. Transpose layers enclose the subtree flattening branch and their output is used in the final part of the network.

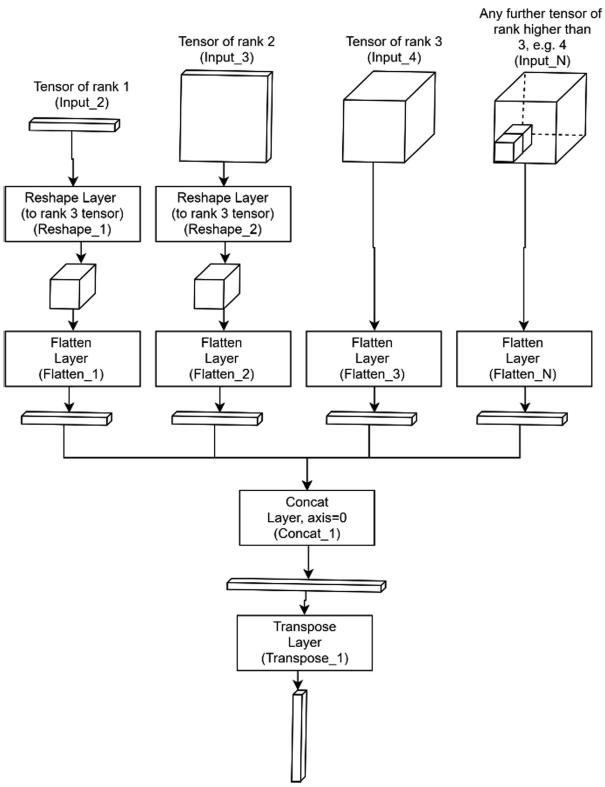

Fig. 4. Subtree flattening branch of the proposed architecture

ShiftMatrix Layers. Those layers are part of the role propagating branch (Fig. 5). The primary and only purpose of this layer is production of the shift matrix that was discussed in Section "Theoretical method of building shift matrix". In practice it is a tensor of rank 2 or an ordinary matrix. It is interesting to estimate it dimensions. Width

of the matrix or a shift operator equals to the size of the vector representing the tree that should be assigned to a given role while height of the matrix equals the size of vector representing a structure assigned to a new role.

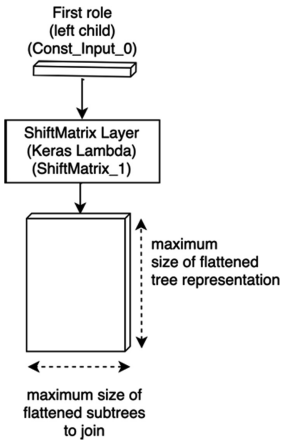

Fig. 5. Role propagating branch

MulVec Layers. Those layers are part of the neural network tail (Fig. 3). Those layers perform ordinary matrix-vector multiplication and the resulting vector contains tensor representation of the current subtree assigned to a new role.

Add Layer. This layer is an output of the network (Fig. 3). All the subtrees are now assigned to new roles and it is required to join them together and the sum vector would represent the resulting structure after joining all subtrees on the tensor level.

5 Conclusion

The novel neural architecture that solved a task of joining structures was proposed and implemented in the Keras framework. The implementation is open-source and available online[2]. Several conceptual gaps of original works devoted to the same topic were closed, in particular the mechanics of building the shift matrix. The elaborated network is robust and is designed to work with arbitrary number of roles and existing tensor representations of different depth. This result provides an essential brick in the bridge between symbolic and sub-symbolic levels of computations.

However, there is still an opened question on performing other operations over arbitrary structures on the tensor level, for example adding or removing nodes or moving nodes to other positions in the structure. Also, current proposal requires initial definition of the structure maximum depth that can be an obstacle in edge cases, as well

[2] https://github.com/demid5111/ldss-tensor-structures.

as constructing the shifting matrix depending on number of roles. So, there is an actual direction for further development of Tensor Product Variable Binding methods.

References

1. Rumelhart, D.E., Hinton, G.E., McClelland, J.L.: A general framework for parallel distributed processing. Parallel Distrib. Process. Explor. Microstruct. Cogn. **1**, 26 (1986)
2. Rumelhart, D.E., McClelland, J.L.: PDP Research Group: Parallel Distributed Processing, 1st edn, p. 184. MIT press, Cambridge (1988)
3. Serafini, L., Garcez, A.D.A.: Logic tensor networks: deep learning and logical reasoning from data and knowledge. arXiv preprint. arXiv:1606.04422 (2016)
4. Teso, S., Sebastiani, R., Passerini, A.: Structured learning modulo theories. Artif. Intell. **244**, 166–187 (2017)
5. Browne, A., Sun, R.: Connectionist inference models. Neural Netw. **14**(10), 1331–1355 (2001)
6. Smolensky, P.: Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artif. Intell. **46**(1), 159–216 (1990)
7. Gallant, S.I., Okaywe, T.W.: Representing objects, relations, and sequences. Neural Comput. **25**(8), 2038–2078 (2013)
8. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196 (2014)
9. Demidovskij, A.: Considering selected aspects of tensor product variable binding in connectionist systems. In: Proceedings of the 2019 Intelligent Systems Conference (IntelliSys). The conference will be held in September, pp. 5–6. Springer, Cham (2019)
10. Smolensky, P., Legendre, G.: The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar (Cognitive Architecture), 1st edn. MIT press, Cambridge (2006)