Implementation Aspects of Tensor Product Variable Binding in Connectionist Systems

Alexander Demidovskij(✉)

National Research University Higher School of Economics,
Bolshaya Pecherskaya Street 25/12, Nizhny Novgorod, Russia
ademidovskij@hse.ru
https://www.hse.ru/staff/demidovs

Abstract. Tensor Product Variable Binding is an important aspect of building the bridge between the connectionist approach and the symbolic paradigm. It can be used to represent recursive structures in the tensor form that is an acceptable form for neural networks that are highly distributed in nature and, therefore, promise computational benefits from using it. However, practical aspects of tensor binding implementation using modern neural frameworks are not covered in the public research. In this work, we have made an attempt of building the topology that can perform binding operation for a well-known framework Keras. Also we make the analysis of the proposed solution in terms of its applicability for other important connectionist aspects of Tensor Product Variabl Binding. Proposed design of the binding network is the first step towards expressing any symbolic structure and operation in the neural form. This will make it possible for traditional decision making algorithms to be replaced with a neural network that brings scalability, robustness and guaranteed performance.

Keywords: Connectivism · Decision support systems · Tensor computations · Neural networks · Unsupervised learning

1 Introduction

Any knowledge should be expressed in the form of formalized structures in order to be used in mathematical models and computations. The question on how to express that knowledge is the key in every symbolic and sub-symbolic or connectivist approach.

With a certain simplification, we can think of symbolic approach as representing traditional algorithms when we operate with understandable structures, terms and operands. For example, multi-round decision making, auction algorithms and so on. At each step of symbolic computation we can understand the intermediate results as they are symbolic structures.

© Springer Nature Switzerland AG 2020
Y. Bi et al. (Eds.): IntelliSys 2019, AISC 1037, pp. 97–110, 2020.
https://doi.org/10.1007/978-3-030-29516-5_9

With the same simplification, sub-symbolic or connectivist approach [1] can be described, for example, by neural computations during which multiple tensors (matrices, vectors) are created and absolutely meaningless until we get the final result. There is no place for symbolic structures in this paradigm. It is especially important when we want to build the bridge between connectivist and symbolic paradigms. This transition should have a sort of communication protocol and therefore formal knowledge plays that role.

In the decision making field we are trying to find the solution based on the problem knowledge. Usually it is kept from stakeholders who share that knowledge with us. One of the possible ways to express this knowledge is to use ontologies [2]. However, regardless the way we express the knowledge we need to create the hierarchy. Hierarchy is required to represent relations between elements, otherwise we loose the semantics imposed by the way those elements interact. In general, such a structure has unbound nesting levels that considerably complicates creation of representation in a vector format - a natural input for neural networks.

One of simple examples of such an hierarchy is the natural language sentence parsing [3]. Hierarchy is essentially built in the sentence in the way how words are used there and how they are connected. For example, adjective mandatory relates to a noun, adverb - to a verb. However, when there are several adjectives the only way to understand what adjective relates to what noun is to take a look at the interconnection between them. There are multiple examples of natural language sentence parsing with the help of hierarchy [4], where language is defined as a text and text is processed as a set of sentences (sequences), each of them is used for structure construction.

To sum up, the built hierarchy can be considered as an output of the task solving on the symbolic level. However, there are problems of interpreting this structure in the vector format, because at the end we want to preform computations in the neural network that accepts only numbers as inputs. This is the place where Tensor Product Variable Binding starts shining [1]. We will describe key aspects later in the paper as well as describing technical aspects of building a working network with the modern means of network construction. To our best knowledge this is the first attempt of applying Tensor Product Variable Binding ideas in the modern neural network frameworks.

The structure of the paper is as follows. Section 2 covers the problem overview, including analysis of the existing solutions, Sect. 3 contains an example of applying Tensor Product Variable Binding to a simple structure. After that, in Sect. 4 we describe a high level architecture of the Tensor Product Variable Binding network. Then, in Sect. 5 we describe the proposed architecture of the neural network that solves the binding task. Finally, in Sect. 6 we make conclusions and define directions of further research.

2 Problem Overview

As we have already noted, the transition between symbolic and connectionist paradigms is a key component to make the integration of them possible.

This transition can be made with the means of the First-Order Logic (FOL). Given that expressions from the FOL can be translated to some distributed subsymbolic representation, we can translate our knowledge to the set of expressions in FOL and then, using the known translation rules, transform those expressions in the vector representation [5–7].

There is a huge list of different logics:

1. First-Order Logic [5]. It is important to note that the distributed representation is created from FOL expressions, that in turn, contain predicates and variables.
2. Fuzzy Evidential Logic [6]. It is built around following elements: facts, rules, weighting scheme and conclusion. Moreover, the expression is mandatory split into left and the right part.
3. Probabilistic Soft Logic [7]. In this type of logic it is expected that users define the family of theories that extend the logic, for example, to work with floating point, binary and complex values. Such theories are called Modulo Theories.

As we can see this transition is possible and it plays the critical role in the whole flow to work. In particular, we have the problem situation, then we collect data about it from stakeholders and experts, then we need to somehow translate it to the structure and then get its vector representation, so that we can use it as an input to the neural network. By that we can connect the world of concepts and terms and the world of numbers and activations. One of the ways for building it is the knowledge fitting mechanism [8].

Neural Network is capable of building associations that construct definite structures on the base of the training dataset. However, one of the most profound approaches to representing the structures in the vector format is the Tensor Product Variable Binding approach proposed by Smolensky [1] and further described in the [9].

2.1 Tensor Product Variable Binding

The main idea of Tensor Product Variable Binding [1] is representing the data in the hierarchical form containing elements of two types: role and filler.

Definition 1. *Filler: Element of the structure that is characterized by the role that it plays in the structure.*

Definition 2. *Role: An action, function, model that the given element (filler) plays in the structure.*

Definition 3. *Binding: A connection between a filler and a role representing their relationship in the structure.*

If we define a filler and a role in some space, we can use the rules of tensor algebra for translating the structure in some vector representation.

Therefore, we consider the structure as a system that consists of pairs {role, filler} and each pair is represented as a tensor multiplication of the corresponding pair of vectors.

More importantly, there are local and distributed representation. Local representations are so called "hot-encodings", where each element is represented as a vector that contains all zeros but the single one on some position. Distributed representations are, in turn, arbitrary vectors. In a case we get the new element in the structure and all elements have local representation, we need to add the new dimension to all elements in the structure with zero value in order to keep orthogonality of the representation. For the distributed representation, the number of dimensions is kept the same. Distributiveness can be on the full space or in some sub-space [6].

It is important to note that it is possible to perform the unbinding procedure that is by definition getting of the original role or the filler vector by the given tensor representation of the structure and the filler or role that the element played in the structure. In other words, we can say what role a particular filler played in the structure or what filler played the particular role. This is performed with the usage of the bind space. Finally, it is possible to perform operations over the structure with only using its tensor representation. This opens the door for potential computation of those manipulations with neural networks.

Tensor Product Variable Binding is not the only one way to get the distributed representation of the structure. There also such methods as: Holograhic Reduced Representations (HRRs), Binary Spater Codes and so on [6]. Tensor Product Variable Binding is in the focus of current research due to closeness of its ideas to modern frameworks, therefore, we leave discussion of alternative structure representation solutions out of the scope of this paper.

The general flexibility of the Tensor Product Variable Binding inspired creation of Vector Symbolic Architectures (VSA) [9]. There are three consecutive stages in VSAs:

1. Pre-processing. During this step, symbolic representation (structure) is translated to the sub-symbolic (vector) one. This step is usually performed once to generate required vectors. For example, with the use of embeddings - special vectors that represent words while saving the semantics closed in the context where those words were used [10,11].
2. Sequence generation.
3. Prediction (output calculation).

2.2 Distributed Representations in Connectivist Approach

Distributed representation plays a key role in the sub-symbolic computations. For example, any neural network can not work with anything but tensors [12]. Moreover, in this research we focus on such data which structure is of the same importance as separate elements. There are various methods of getting distributed representations from the natural language data that essentially has a structure: grounding [5], doc2vec [13] and word2vec [14].

We would like to draw attention to the mandatory inclusion of coders and encoders in any workflow that contains sub-symbolic computations [15]. The role of the encoder is transforming the symbolic phrase to the vector form and the role of decoder is in contrary translation of given vector representations in the symbolic phrases. The main difficulty with distributed representations is that they should store not only content of the original sequence (for example, a sentence) but also the structure included in that sequence. There are multiple cases when distributed representations are used for solving the real life cases, for example in Predication-based Semantic Indexing [16].

Usually, distributed representations have a huge dimension and this dramatically increases the computational complexity of the overall solution. There are multiple ways to reduce dimensionality, for example Singular Value Decomposition. For further analysis of the vector space, Hierarchical Clustering is usually used [17]. Finally, there is the recent effort to introduce new entity called Semantic vector that is designed to be a unified part of any solutions working with the distributed representations [16].

To sum up, comparing the connectivist and symbolic levels we can say that symbolic computations are serial and discrete and operations are performed in the user-friendly format - in known structures with known properties. At the same time, connectivist calculations are made over local or distributed representations [8]. Moreover, symbolic level hugely depends on the domain knowledge while neural computations, obviously, do not depend on that [18]. Interestingly, there are some works also devoted to the usage of multi-layer (deep) tensor topologies [19]. However, we strongly consider that there are still remaining questions that should be addressed. In particular, analysis of the existing theoretical architectures and their adoption details in terms of modern frameworks, applicability of such approaches to a broader set of tasks, for example in building expert systems and decision support systems.

3 Tensor Product Variable Binding Calculation Rules

Tensor Product Variable Binding is a way to transform the symbolic structure into the vector format. According to Smolensky [1] it is built on the simple tensor multiplication operation.

Definition 4. *Tensor multiplication is the operation over two tensors* $\boldsymbol{f}$ *of rank* $\boldsymbol{N}$ *and* $\boldsymbol{r}$ *of rank* $\boldsymbol{M}$ *that results in the new tensor* $\boldsymbol{b}$ *of rank* $\boldsymbol{N+M}$ *so that* $b_{ij} = f_i * r_j$.

Definition 5. *Tensor product* ψ *for the given structure* S *containing pairs* $\{f_i, r_i\}$ *is calculated in the following way:*

$$\psi = \sum_i f_i \bigotimes r_i \tag{1}$$

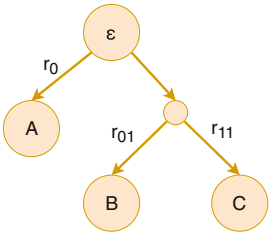


Fig. 1. A simple structure for demonstration of Tensor Product Variable Binding

In order to understand the ground principles for constructing the distributed matrix manipulation mechanism in a form of neural network it is important to understand how the full workflow works for the simple structure (Fig. 1).

Let the structure be a directed acyclic graph with three leaves: A, B, C and the root ε. As we already know the structure is needed to describe relationships between the elements. In the sample structure it is obvious that A is the left child of the root, while B and C are children of the right child of the root. We ignore that right child because it is not a filler and does not bring any value in our structure representation. However, we are still interested in other two fillers B and C and their connections. Having defined fillers is not enough for representing the structure according to the Tensor Product Variable Binding rules because we also need to describe roles. Again, from the diagram we can see that there are three different roles: r_0, r_{01} and r_{11}. Their meaning is quite simple: r_0 denotes the role "left child of the root", r_{01} - the role "the left child of the right child of the root" and r_{11} - means the role "the right child of the right child of the root".

Semantically those rules and fillers are considered different and that should be reflected in the vector representation of those elements of the structure. Then we are free to define the vectors representing fillers matrix F (2) and roles matrix R (3). We need to make sure that they are at least linearly independent or, what is easier for us, orthogonal and normalized.

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3}$$

According to (1) we can translate the given structure to the vector representation by performing pairwise tensor multiplication over the fillers and roles (4) with the final sum of the terms. In particular:

$$
\begin{aligned}
f_1 \bigotimes r_1 &= \begin{bmatrix}1&0&0&0&0\end{bmatrix} \bigotimes \begin{bmatrix}1&0&0&0&0\end{bmatrix} = \begin{bmatrix}1&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix} \\
f_2 \bigotimes r_2 &= \begin{bmatrix}0&1&0&0&0\end{bmatrix} \bigotimes \begin{bmatrix}0&1&0&0&0\end{bmatrix} = \begin{bmatrix}0&0&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix} \\
f_3 \bigotimes r_3 &= \begin{bmatrix}0&0&1&0&0\end{bmatrix} \bigotimes \begin{bmatrix}0&0&1&0&0\end{bmatrix} = \begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&0\\0&0&0&0&0\end{bmatrix}
\end{aligned} \tag{4}
$$

As we have found standalone tensor multiplications for all three pairs of fillers and roles in the structure, we are able to find the final tensor representation of the structure by performing a simple element-wise sum over the given matrices:

$$
\begin{aligned}
\psi &= f_1 \bigotimes r_1 + f_2 \bigotimes r_2 + f_3 \bigotimes r_3 \\
&= \begin{bmatrix}1&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&0\\0&0&0&0&0\end{bmatrix} = \begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&0&0\end{bmatrix}
\end{aligned} \tag{5}
$$

Finally, we get (5) that is the tensor representation of the given structure S (Fig. 1). In the next section we will learn how those operations can be performed in the neural computing paradigm.

4 High-Level Theoretical Architecture for Tensor Product Variable Binding

It was already mentioned above that the Tensor Product Variable Binding was proposed in [1] as well as the network paradigm that can create tensor representation of the given pairs of roles and fillers. This high level architecture is presented in Fig. 2.

We start reviewing this architecture from the case when the network performs one binding operation per the given time period. Figure 2a represents such a network. As we can see the idea is that the network has two inputs: for a filler $\widetilde{f}$ and a role $\widetilde{r}$ vectors correspondingly. The network itself consists of sigma-pi units [20].

Each sigma-pi unit has one or 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 $\{I_{\sigma i}\}$. The resulting value of the unit can be computed as a weighted sum of products from each input site (6).

$$
v = \sum_{\sigma} w_\sigma \times \prod_{i} I_{\sigma i} \tag{6}
$$

However, weights are ignored in the proposed architecture, so they are equal to *1* for each input site and the formula (6) is a bit simplified (7).

$$v = \sum_{\sigma} 1 \times \prod_{i} I_{\sigma i} = \sum_{\sigma} \prod_{i} I_{\sigma i} \tag{7}$$

A well known advantage of neural networks is the high level of computations distributiveness and easy scalability on the growing number of inputs that is often called batching. There is an extension of the network accepting two vectors and therefore performing serial computations to the architecture that provides simultaneous binding operation for N pairs of such vectors. This extension is presented on the Fig. 2b. For simplicity, we refer to the case when the network performs two simultaneous binding operations.

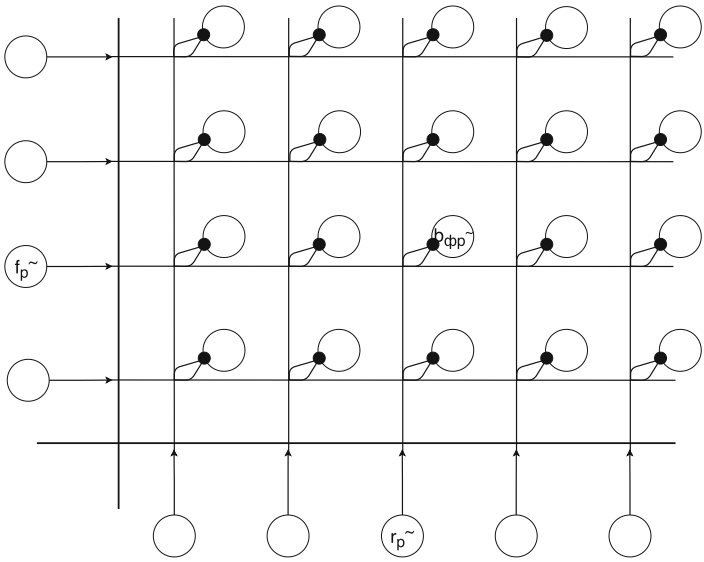
The overall idea of the network that consists of sigma-pi units is kept the same although each sigma-pi unit now contains N input sites. In our case each input site receives signals from the corresponding values in input vectors. The nice fact about this architecture is that adding new inputs and new connections to the sigma-pi units is enough to add parallelism in network computations. In other words we utilize natural properties of neural networks.

However, a gap between the theory and practice exists so that practitioners face the problem of expressing the network architecture in the terms of modern frameworks and approaches to training and inference of neural networks. The purpose of this research is to close this gap for the binding network and we will demonstrate it in the following section.

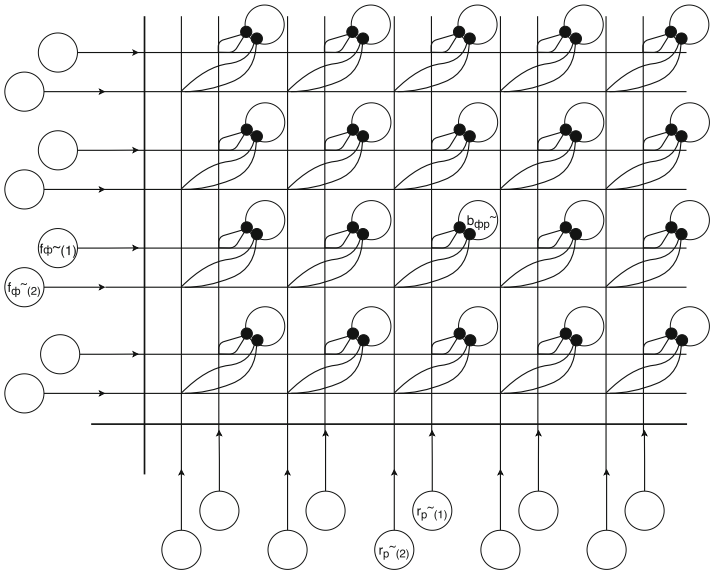
5 Modern Architecture for Tensor Product Variable Binding

As you have seen from the previous section, the proposed network is rather high level and described in very abstract terms. This brings a huge gap when someone decides to use this architecture in enterprise applications or for some further research. Practically, all the questions about the network inputs, outputs, mechanics, implementation details are left not described although they are critical for building real life solutions.

Therefore, in this paper, we propose the implementation of the topology that performs binding operation. You can find the overall scheme in Fig. 3. It is important to mention that we prototype the network in the Keras [21] framework that is the superstructure over the popular frameworks: TensorFlow [22] and PyTorch [23]. The main advantage of this framework is the high level of abstraction for network description when compared to other analogues. This framework is considered to be ideal for prototyping and therefore is our framework of choice.



(a) Network architecture for the serial binding operation



(b) Network architecture for two simultaneous bindings operations

Fig. 2. High-level theoretical architecture for Tensor Product Variable Binding network [1]

5.1 Topology Structure

Network Inputs. As we can see the network accepts two inputs: one for fillers and one for roles. Each input is a batch of vectors while batch can vary depending on the number of roles and fillers in our structure. Using batch is a common approach for building networks that automatically scale for the given number of input samples. Note that roles and fillers are not constrained to be of the same size as in general they represent different spaces. We do not perform any additional manipulations with the input and let the network do it for us. In other words we feed the network with raw data.

Preparing Inputs for Binding. The next step is preparing one of the inputs for tensor multiplications. It is easy to see that tensor multiplication over vectors can be expressed in a usual vector product operation with one of the vectors being transposed (8):

$$v = f_i \bigotimes r_i = f_i^T \times r_i \tag{8}$$

This equivalence lets us avoid using tensor product directly in the network described in the terms of the Keras framework. It is extremely vital as the tensor product layers are absent in all modern frameworks. Therefore we perform the permutation operation that switches dimensions in a way that we can use vector-vector multiplication with standard framework layers.

Vector Multiplication Layer. This brings us to the next layer that has two inputs: raw fillers vectors left unchanged and transposed role vectors. This layer performs the operation described in (8). Although this is the usual vector-vector multiplication it was a surprise for us to recognize absence of a layer that receives two vectors and outputs the product of them.

However, as it was already stated, Keras has a flexible and expandable architecture, therefore we created the custom layer that computes product of two vectors. It is a primitive Lambda layer (Listing 1.1).

Listing 1.1. Implementation of the vector-vector multiplication layer

```
from keras.layers import Lambda
def mul_vec_on_vec(tensors):
  return [tensors[0][i] * tensors[1][i] \
          for i in range(len(fillers))]
binding_cell = Lambda(mul_vec_on_vec)
```

Sum Layer. This layer takes a variable number of input matrices as an input and performs the element-wise sum over them. According to the definition of the tensor product for the given structure (1), we need to perform exactly this operation to get the tensor representation.

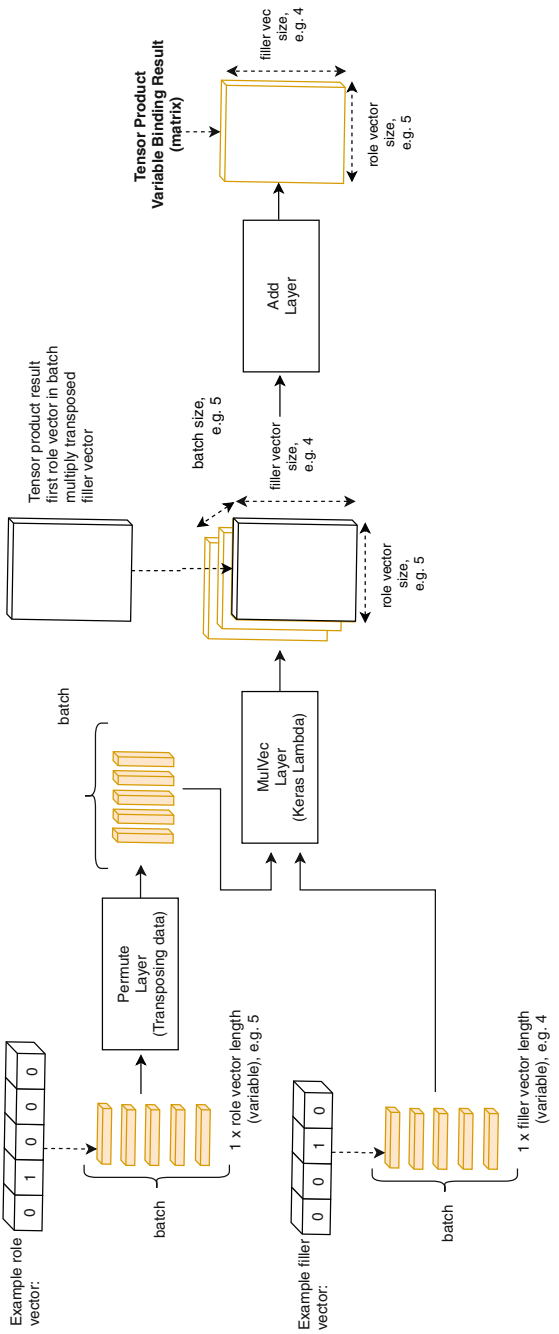


Fig. 3. Modern architecture for Tensor Product Variable Binding

5.2 Training and Inference

Once the topology is designed and described in the terms of the framework, we can move to its training and inference. However, for the given binding network we can see that it is designed for a feed forward inference without preliminary training. In point of fact, the topology does not keep any weights which are usually trained in canonical neural models. Speaking about the inference side, it absolutely follows the example that we examined in Sect. 3.

The source code for further experiments can be found at the open source repository[1].

6 Conclusion

To sum up, we have demonstrated how the binding network can be implemented with the means of the Keras framework, what are the obstacles and limitations of modern frameworks when they are applied to the implementation of such networks. Also we have considered inference aspects.

However, when analysing the proposed architecture, we see the following cons of the design:

1. Absence of training. The network is not trainable due to absence of weights. When it is planned to reuse existing binding values the network should be inferred again to obtain values. Instead, they could be trained and stored in the serialized network file.
2. Absence of any structure manipulation on the tensor level. From the perspective of current research, it is possible to translate a structure to the tensor level, however, it is not clear what we can do with that. There are numerous potential operations that can be done over the structure: adding new children, replacing elements, removing them completely. That is why we should be able to make such transformations not only on the symbolic level, but also on the connectivist side or, in other words, on the tensor level.
3. Need in the decoder. The binding network by definition plays the role of encoder translating the given arbitrary structure to the tensor representation. However, after the structure is encoded and changed on the tensor level, it should be translated to the symbolic form again. Otherwise it is not possible to analyse the result gained by the neural network.

Apart from the fact that this research clearly closes the practitioner gap in implementing binding networks, we can highlight several advantages of such an architecture:

1. Scalability. The network does not depend on the number of fillers and roles. It can accept any quantity of them with the obvious requirement that each filler should have one and only one matching role. Moreover, this scalability is inherently built in the topology with the usage of batch dimension.

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

2. Simplicity. The network does not contain overcomplicated parts that are hard for implementation. For example, sigma-pi units can be expressed by a combination of primitive layers that do the same operations but with clear and transparent data flow. Moreover, using the sigma-pi unit is overcomplication as by definition it contains weights, but in binding network it is ignored (set as equal to 1).
3. Known language for practitioners. The binding topology architecture was proposed 30 years ago and the field has rapidly rocketed from that moment as well as tooling and frameworks. Binding mechanism is a crucial part of Tensor Product Variable Binding mechanism and its derivatives and it is vital to express the topology in modern terms for further development of the field.

To sum up, we consider items 2 and 3 from the list of current design limitations as concrete directions for further research. At the same time we formulate following research questions that are still open:

1. Training in sub-symbolic systems. Do we need it to be a supervised or unsupervised one? Do we need a labelled data or such networks can generalize and learn patterns from raw data without a teacher?
2. Attainability of any symbolic operation expression in terms of neural paradigm so that the neural model produces results acceptable by accuracy? What about expressing traditional decision making approaches in the neural form? A positive answer could give a start for huge usage of connectivist ideas in dozens of actual problems including those from the decision making domain.

References

1. Smolensky, P.: Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artif. Intell. **46**(1–2), 159–216 (1990)
2. Wang, H., Dou, D., Lowd, D.: Ontology-based deep restricted Boltzmann machine. In: International Conference on Database and Expert Systems Applications, pp. 431–445. Springer (2016)
3. Margem, M., Yilmaz, O.: How much computation and distributedness is needed in sequence learning tasks? In: Artificial General Intelligence, pp. 274–283. Springer (2016)
4. Dehaene, S., Meyniel, F., Wacongne, C., Wang, L., Pallier, C.: The neural representation of sequences: from transition probabilities to algebraic patterns and linguistic trees. Neuron **88**(1), 2–19 (2015)
5. Serafini, L., d'Avila Garcez, A.: Logic tensor networks: deep learning and logical reasoning from data and knowledge. arXiv preprint arXiv:1606.04422 (2016)
6. Browne, A., Sun, R.: Connectionist inference models. Neural Netw. **14**(10), 1331–1355 (2001)
7. Teso, S., Sebastiani, R., Passerini, A.: Structured learning modulo theories. Artif. Intell. **244**, 166–187 (2017)

8. Besold, T.R., Kühnberger, K.-U.: Towards integrated neural-symbolic systems for human-level AI: two research programs helping to bridge the gaps. Biol. Inspired Cogn. Archit. **14**, 97–110 (2015)
9. Gallant, S.I., Okaywe, T.W.: Representing objects, relations, and sequences. Neural Comput. **25**(8), 2038–2078 (2013)
10. Blacoe, W., Lapata, M.: A comparison of vector-based representations for semantic composition. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 546–556. Association for Computational Linguistics (2012)
11. Cheng, J., Wang, Z., Wen, J.-R., Yan, J., Chen, Z.: Contextual text understanding in distributional semantic space. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 133–142. ACM (2015)
12. Rumelhart, D.E., McClelland, J.L., PDP Research Group, et al.: Parallel Distributed Processing, vol. 1. MIT Press, Cambridge (1987)
13. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196 (2014)
14. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
15. Shang, L., Lu, Z., Li, H.: Neural responding machine for short-text conversation. arXiv preprint arXiv:1503.02364 (2015)
16. Widdows, D., Cohen, T.: Reasoning with vectors: a continuous model for fast robust inference. Log. J. IGPL **23**(2), 141–173 (2014)
17. Frank, R., Mathis, D., Badecker, W.: The acquisition of anaphora by simple recurrent networks. Lang. Acquis. **20**(3), 181–227 (2013)
18. Yilmaz,Ö., d'Avila Garcez, A.S., Silver, D.L.: A proposal for common dataset in neural-symbolic reasoning studies. In: NeSy@HLAI (2016)
19. Wang, H.: Semantic deep learning, pp. 1–42. University of Oregon (2015)
20. Rumelhart, D.E., Hinton, G.E., McClelland, J.L., et al.: A general framework for parallel distributed processing. Explor. Microstruct. Cogn. **1**(45–76), 26 (1986)
21. Chollet, F., et al.: Keras (2015). https://keras.io
22. 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., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). Software available from tensorflow.org
23. Paszke, A., Gross, S, Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)