Designing arithmetic neural primitive for sub-symbolic aggregation of linguistic assessments

Journal of Physics: Conference Series **1680** (2020) 012007 doi:10.1088/1742-6596/1680/1/012007

Designing arithmetic neural primitive for sub-symbolic aggregation of linguistic assessments

Alexander Demidovskij and Eduard Babkin
ul. Bolshaya Pecherskaya 25/12, Nizhny Novgorod 603005, Russia
E-mail: ademidovskij@hse.ru, eababkin@hse.ru

Abstract. The very first step towards a challenging goal of creation of monolithic generic neuro-symbolic systems is application of sub-symbolic ideas to particular symbolic algorithms like aggregation of fuzzy linguistic assessments during Linguistic Decision Making. A novel theoretical idea is to express this aggregation as structural manipulations and translate them in a neuroalgorithm. Tensor Product Representation (TPR) methodology provides a generic framework of designing neural networks that do not require training and produce an exact result equivalent to the result of symbolic algorithms. This paper demonstrates design of TPR-based arithmetic as a basic building block for expressing linguistic assessments aggregation on a sub-symbolic level and a neural architecture for the basic arithmetic operation.

1. Introduction

Building of integrated monolithic neural-symbolic systems is an actual task of modern computer science field [1], [2], [3] because it can help in understanding human cognition. Cognition operates with complex symbolic data structures: graphs, trees, shapes, grammars etc., performs symbolic manipulations with means of symbolic logic. At the same time, processing of these structures in mind is performed on the neural level. Artificial Neural Networks (ANN) operate on the sub-symbolic level as a distributed computational mechanism. There is an important scientific task of constructing neural networks that perform significant intellectual tasks without preliminary training stage [4] in massively parallel computation environments like multi-agent systems or Internet Of Things (IoT) [5].

Multi-criteria Linguistic Decision Making is huge field that tries to extend existing decision making methods with fuzzy assessments from experts that are mostly expressed in a linguistic format [6], [7]. Designing decision support systems as neural-symbolic solution is an actual, challenging and open task [4], [8]. The very first step towards this goal could be translation of linguistic assessments aggregation to a neural level. By design this aggregation consists of three steps: encoding assessments as numbers, arithmetic manipulations with those numbers and decoding final assessment to a linguistic form [9]. There is an ongoing research [10], [11] demonstrating that structural manipulations can be successfully expressed in a neural form. Therefore, if non-negative integer arithmetics could be expressed as structural manipulations, then, theoretically, linguistic aggregation could happen on the neural level. This paper is aimed at proposing a design of a neural network that is capable of operating over structures that represent numbers and that performs simple arithmetic operation of increasing a given

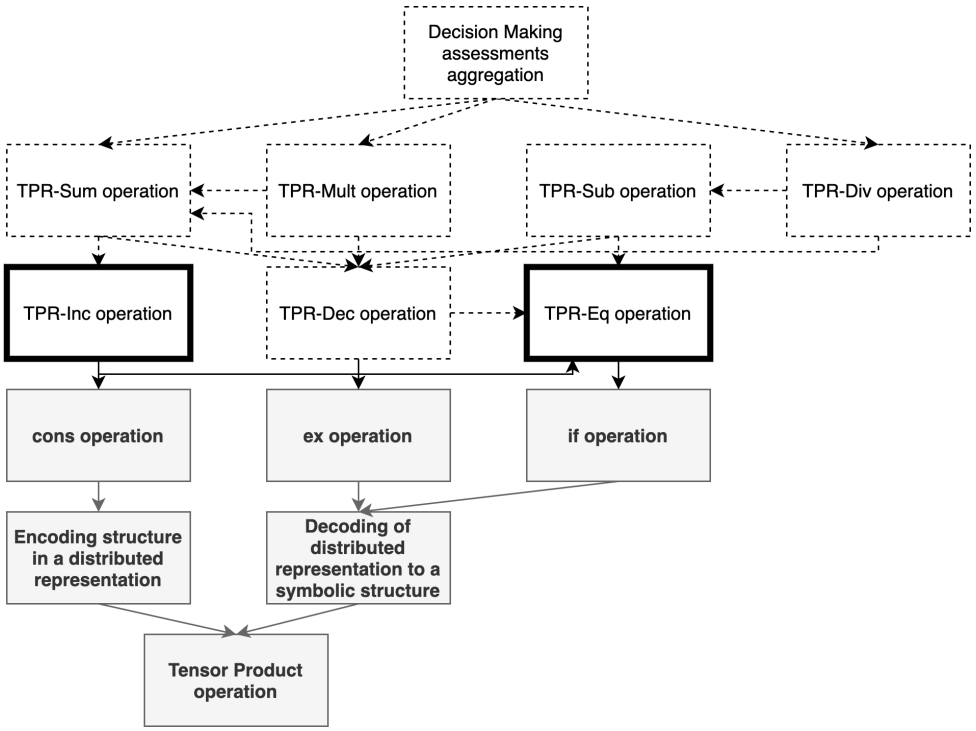

Figure 1. Evolution of sub-symbolic methods required for expression of linguistic assessments aggregation during decision making process. Grayed cells refer to the existing research, thick-bordered cells represent current research direction, dashed-bordered cells stand for directions of further research.

number by one. From the standpoint of the overall research direction, this work is a continuing contribution to the idea of sub-symbolic aggregation of linguistic assessments (Fig. 1).

The article is structured as follows. Section 2 gives a short overview of methods that allow encoding of symbolic structures. Section 3 demonstrates basic arithmetic operations expressed as manipulations with structures. Neural design of the primitive that performs incrementing operation is proposed in Section 4. Final remarks and directions of further research conclude the article.

2. Background study

There are multiple ways of transforming the recursive structure in a distributed format. One of them is Tensor Product Variable Binding (TPVB) proposed by Paul Smolensky in [12]. This approach became a foundation for other methods, like Holographic Reduced Representations (HRRs), Binary Spater Codes [13] and Vector Symbolic Architectures (VSA) [3]. There multiple methods proposed in research of Knowledge Base (KB) translation to a form of First-Order Logics (FOLs) with strictly defined rules and further encoding of such expressions [14], [15], [16].

Tensor Product Variable Binding (TPVB) as a binding mechanism that encodes arbitrary symbolic structures in a distributed representation and symbolic operations as weights of the network so that the resulting structure emerges as a network inference result. TPVB allows to build recursively such distributed representations on top of atomic elements: fillers and roles.

Definition 1 *Fillers and roles [12]. 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:*

$$\psi = \sum_i f_i \bigotimes r_i \tag{2}$$

Given notation of roles and fillers, that should be encoded in a linear independent manner, there are two primitive operations defined: *cons* and *ex*. As it was proved in [17] 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:*

$$\begin{aligned} cons(p, q) &= p \bigotimes r_0 + q \bigotimes r_1 \\ &= W_{cons_0} p + W_{cons_1} q \end{aligned} \tag{3}$$

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:*

$$W_{cons_0} = I \bigotimes 1_A \bigotimes r_0, \tag{4}$$

where I is the identity matrix on the total role vector space, 1_A is the identity matrix $A \times A$. W_{cons} matrices are defined in the manner similar to the W_{cons} matrices that join two sub-trees in one structure [17], [11]. Extraction operation ex is defined in analogous way, 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. This operation is used in both extraction branches that are described in Section 4.

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

$$ex_0(s) = ex_0(cons(p, q)) = p, \tag{5}$$

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:*

$$W_{ex_0} = I \bigotimes 1_A \bigotimes u_0, \tag{6}$$

Operations *cons*, ex_0, ex_1 are equivalent to operations over lists in software general-purpose functional programming languages like Lisp: *cons*, *car*, *cdr*. Therefore, implementation of these operations on the neural level opens the horizon for neural-symbolic computation of symbolic algorithms such as linguistic assessments aggregation during the decision making process.

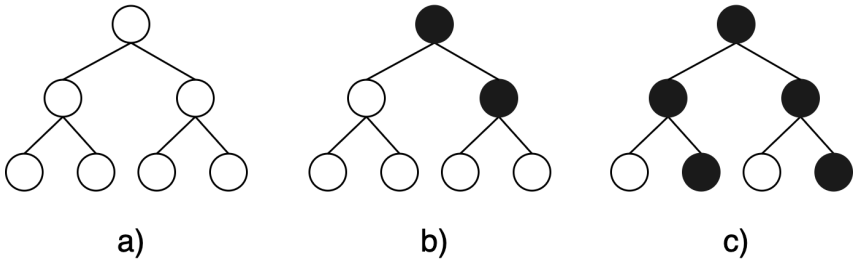

Figure 2. Representing non-negative integers as structures. a) a structure representing $\{0\}$ b) a structure representing 1 c) a structure representing 2. Note: depth of the tree depends on the maximum value of integer.

3. Design of TPR-based arithmetic as a foundation for 2-tuple aggregation

Aggregation of linguistic information implies translation to the numeric values according to [9]. This section demonstrates how basic arithmetic rules can be expressed in terms of structural manipulations with the help of TPR. To design TPR-based arithmetic the axiomatic should be defined, including both primitives and operators. It was decided that the best procedure would be to formulate arithmetic in a Peano arithmetic [18] manner, where three basic actions should be defined: *inc* - increase by one, *equal* - check for equality and *if* – select between actions depending on equality of condition to zero. In the framework of that axiomatic and following basic TPR principles we propose to consider any number as a tree with two positional roles: r_0 (left child) and r_1 (right child). As it was mentioned above, there are two primitive values: 0 (Fig. 1.a), 1 (Fig. 1.b). Those two primitive values are used for construction of other numbers, for example 2 (Fig. 1.c). Using the structures proposed basic arithmetic operations can be expressed as follows.

Definition 7 *TPR-Inc. The operator for increasing a value by one. Let 1 denote a structure, representing integer 1. Then incrementing operator receives a structure a, representing a number as an input:*

$$inc(a) = cons(a, \{1\}). \tag{7}$$

Definition 8 *TPR-Dec. The operator for decreasing a value by one. Let $\{0\}$ denote a structure, representing 0. Then decrementing operator receives a structure a, representing a number, as an input:*

$$dec(a) = \begin{cases} \{0\} & if equal(a, \{0\}) \\ ex_0(a) & otherwise \end{cases} \tag{8}$$

Definition 9 *TPR-Eq. The operator for checking two structures on equality. Let a, b denote two structures. Then the operator can be defined by:*

$$equal(a, b) = \begin{cases} equal(dec(a), dec(b)) & if a \neq \{0\}, b \neq \{0\} \\ \{0\} & if a \neq \{0\}, b = \{0\} \\ \{0\} & if a = \{0\}, b \neq \{0\} \\ \{1\} & if a = \{0\}, b = \{0\} \end{cases} \tag{9}$$

Definition 10 *TPR-Sum. The operator for sum of two integers. Let a, b denote two structures. Then the operator can be defined by:*

$$plus(a, b) = \begin{cases} plus(dec(a), inc(b)) & if a \neq \{0\} \\ b & if a = \{0\} \end{cases} \tag{10}$$

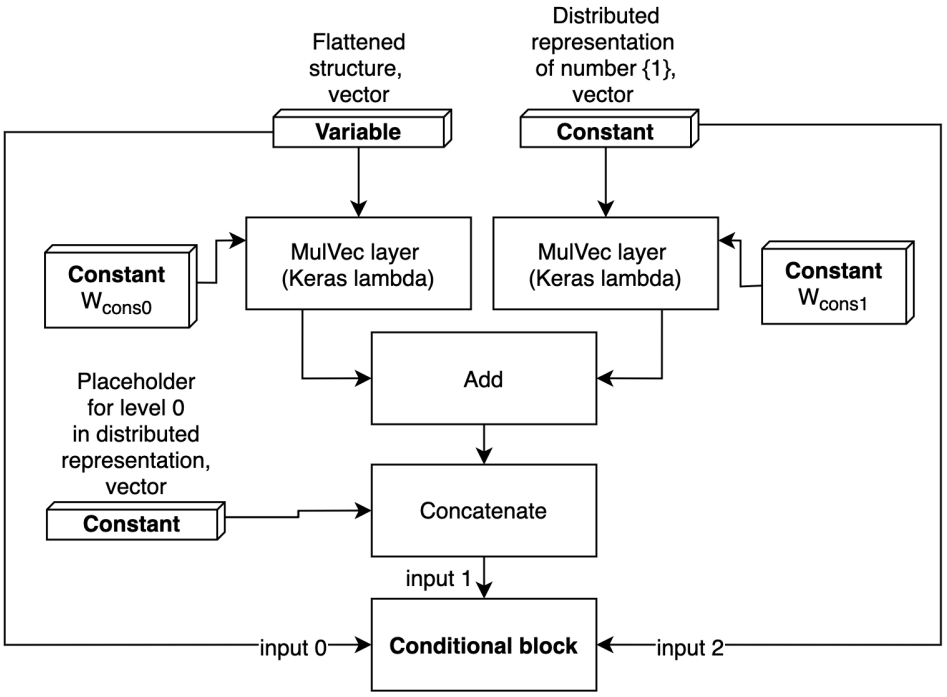

Figure 3. Proposed architecture of TPR-Inc Network.

4. Proposed neural design of TPR-Inc Network

From definitions of basic arithmetic operations in the previous section and the overall scheme of achieving sub-symbolic aggregation of linguistic assessments (Fig. 1) it is clear that *TPR-Inc* is one of main building blocks. This paper aims at proposing a design of a neural network that performs incrementing operation over the distributed representation of a number.

A neural network accepts one variable input and 6 constant inputs (Fig. 3)[1]. A variable input is a flattened distributed representation of a structure that in turn is encoded number. Neural network can be re-compiled for a representation of arbitrary size. As for the constant inputs, the most considerable one is distributed representation of number 1 as it is used for incrementing the input number. Other constant inputs will be discussed below.

Neural network inference starts from the *cons* operation expressed via parallel application of role r_0 to the input structure and role r_1 to the constant input 1. Application of roles is performed with the help of W_{cons_0} and W_{cons_1} matrices (4). After the roles are applied, parallel branches are merged and the output represents a structure that is likely to stand for an incremented number. The joining procedure is explained in huge details in [11]. Due to specifics of such a neural implementation, the resulting tensor should be concatenated with a simple constant stub in order to make it correctly decoded by the existing mechanism described in [10]. Then the execution is transferred to the *Conditional Block*.

4.0.1. Conditional block Conditional block is needed to track the case when the input number is 0. In that case, aforementioned joining procedure of input number and a constant 1 would not produce a distributed representation of 1, although mathematically it is 1. It is explained by the fact that we have two primitive values defined and any computation in terms of TPR-based arithemtics should handle this case. Conditional block accepts three variable inputs and

[1] From now on network description contains terminology accepted in the Keras [19] and TensorFlow [20] software frameworks

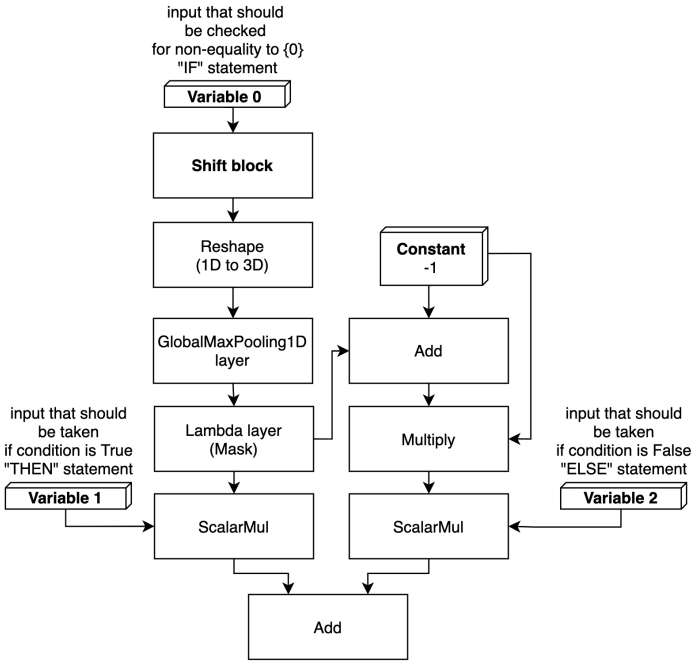

Figure 4. Architecture of condition block.

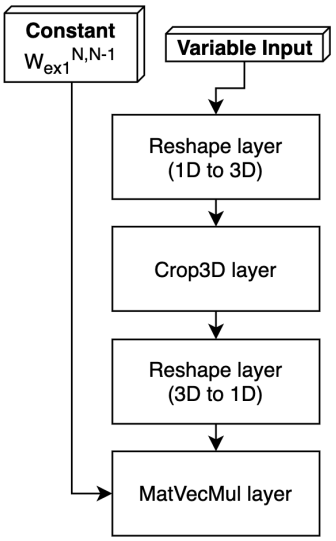

Figure 5. Architecture of shift block.

one constant input. Variable inputs are the tensor that should be checked on equality to 0, the second input stores the tensor that should be taken as an output of the block in case the first variable is bigger than zero, the third input stores tensor that should be taken as output otherwise.

Conditional block inference starts from the Shift Block. In turn, Shift Block receives one variable input that is the structure that should be used for extraction of its child and one constant input that stores the W_{ex_1} matrix (6). Shift block reshapes the input vector in the three dimensional tensor with two dimensions equal to one. This is a requirement from the Keras framework in order to use Crop layer. Crop layer is used to adjust the tensor size for the extraction procedure that is the final operation of the whole block.

Once the shift block extracts the right child of the input structure of the *Conditional Block*, this child is then checked with a sequence of Reshape, GlobalMaxPooling1D and Mask layers, that perform check of the input tensor on the non-equality to 0 and result in a single binary scalar value. It is used as a multiplier for the second variable input of the *Conditional Block* and as an input for the sequence of layers that negate this multiplier and apply the result to the third variable input. Finally, both parallel branches are merged, while it is guaranteed that at least one branch is absolutely filled with zeros. This conditional block is of high reuse for future networks that implement other arithmetic procedures from Fig. 1 and Section 3.

5. Conclusion

It was demonstrated that numbers can be encoded as a recursive TPR-based structure and integer arithmetic can be expressed in a form of structural TPR-based manipulations. Moreover, the basic operation of incrementing an integer value is designed in a neural form. Implementation of the neural network primitive is available as an open-source project[2]. This is proof of concept that demonstrates feasibility of building a neural-symbolic system for aggregation of

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

linguistic assessments. The further direction of research is designing small neural models for other arithmetic operations on top of the proposed incrementing architecture. The techniques described in this article may be used to build networks that perform fast and reliable linguistic assessments aggregation.

Acknowledgments
The reported study was funded by RFBR, project number 19-37-90058.

References
[1] Besold T R *et al.* 2017 *arXiv preprint arXiv:1711.03902*
[2] Besold T R and Kühnberger K U 2015 *Biologically Inspired Cognitive Architectures* **14** 97–110
[3] Gallant S I and Okaywe T W 2013 *Neural computation* **25** 2038–2078
[4] Pinkas G, Lima P and Cohen S 2013 *Biologically Inspired Cognitive Architectures* **6** 87–95
[5] Yousefpour A, Nguyen B Q, Devic S, Wang G, Kreidieh A, Lobel H, Bayen A M and Jue J P 2020 *arXiv preprint arXiv:2002.07386*
[6] Cheng P, Zhou B, Chen Z and Tan J 2017 *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)* (IEEE) pp 1603–1607
[7] Demidovskij A and Babkin E 2019 *Business Informatics* **13**
[8] Golmohammadi D 2011 *International Journal of Production Economics* **131** 490–504
[9] Wei C and Liao H 2016 *International Journal of Intelligent Systems* **31** 612–634
[10] Demidovskij A 2019 *Proceedings of the 2019 Intelligent Systems Conference (IntelliSys)* (Association for Computational Linguistics)
[11] Demidovskij A V 2019 *International Conference on Neuroinformatics* (Springer) pp 375–383
[12] Smolensky P 1990 *Artificial intelligence* **46** 159–216
[13] Browne A and Sun R 2001 *Neural Networks* **14** 1331–1355
[14] Serafini L and Garcez A d 2016 *arXiv preprint arXiv:1606.04422*
[15] Teso S, Sebastiani R and Passerini A 2017 *Artificial Intelligence* **244** 166–187
[16] Pinkas G, Lima P and Cohen S 2012 *International Conference on Artificial Neural Networks* (Springer) pp 482–490
[17] Smolensky P and Legendre G 2006 *The harmonic mind: From neural computation to optimality-theoretic grammar (Cognitive architecture), Vol. 1* (MIT press)
[18] Van Heijenoort J 1967 *From Frege to Gödel: a source book in mathematical logic, 1879-1932* (Harvard University Press)
[19] Chollet F *et al.* 2015 Keras `https://keras.io`
[20] Martin A *et al.* 2015 TensorFlow: Large-scale machine learning on heterogeneous systems software available from tensorflow.org URL `http://tensorflow.org/`