TP-N2F: Tensor Product Representation for Natural To Formal Language Generation

Kezhen Chen*
Northwestern University
Evanston, IL 60201
kzchen@u.northwestern.edu

Qiuyuan Huang
Microsoft Research
Redmond, WA 98052
qihua@microsoft.com

Hamid Palangi
Microsoft Research
Redmond, WA 98052
hpalangi@microsoft.com

Paul Smolensky
Microsoft Research
Redmond, WA 98052
psmo@microsoft.com

Kenneth D. Forbus
Northwestern University
Evanston, IL 60201
forbus@northwestern.edu

Jianfeng Gao
Microsoft Research
Redmond, WA 98052
jfgao@microsoft.com

Abstract

Producing formal-language, especially like relational representations, from natural-language input is crucial for many important reasoning tasks such as program synthesis, problem solving or semantic parsing. In relational representations like programs, mathematical expressions, and ontological knowledge base, discrete *structure* is crucial. The unconstrained vector representations used in typical state-of-the-art deep-learning sequence-processing models such as LSTMs do not explicitly capture such structure. In this paper, we present a new Seq2Seq model for natural-language to formal-language mapping, ***TP-N2F***, based on *Tensor-Product Representations (TPRs)* for embedding symbolic structure in vectors. The TP-N2F encoder employs TPR-binding (an outer product) to create a structured encoding of the entire input and the decoder deploys TPR-unbinding (an inner product) to generate one formal expression at a time: a relational tuples consisting of a relation (or operation) and its arguments. TP-N2F considerably outperforms LSTM-based Seq2Seq models on two different tasks, operation-sequence generation to solve math problems and Lisp-program generation. Our model achieves state-of-the-art performance on the MathQA dataset and the AlgoLisp dataset. Ablation studies show that both the TPR-binding in the encoder and the TPR-unbinding in the decoder contribute importantly to the performance improvement. Overall the experiments suggest that TP-N2F models based on TPR theory can achieve significant structure-learning ability on symbolic reasoning tasks.

1 Introduction

When people perform symbolic reasoning or pattern recognition, they can easily describe the way to the conclusion step by step via relational descriptions. There is evidence that relational representations are important for human cognition (1; 2). Several researches also show that neural-symbolic structural representations are very efficient for both language and vision tasks (3; 4; 5). As the rapid development of artificial intelligence, a growing number of researchers tend to use deep learning models to solve complicated symbolic reasoning tasks and language tasks. However, most existing deep learning sequence models such as LSTMs do not explicitly capture the human-like relational structure information as required.

*Kezhen worked on this project during the internship at Microsoft Research.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

In this work, we propose a novel neural architecture - ***TP-N2F*** - to solve natural- to formal- language generation task (N2F). TP-N2F encodes natural language symbolic structure in vector-space and decode it as relational representations over Tensor Product Representation (TPRs) (6) method. TPRs can encode complex symbolic structures as vector-space embeddings using TPR-binding (7) and disentangle the symbolic structures from vectors using TPR-unbinding (8; 9). In this paper, we use both TPRs to represent a symbolic natural-language scheme in vector-space and generate relational representations via TPR-binding and TPR-unbinding.

Our contributions in this work as follows. Firstly, we propose a new TP-N2F scheme to learn symbolic structure mapping from natural- to formal- language in vector-space. Secondly, we present a new Seq2Seq TP-N2F model for N2F task. Based on our knowledge, this is the first model be proposed which combines both binding and unbinding features of TPRs to achieve generation tasks in deep learning. Finally, state-of-the-art performance on two challenge tasks shows that TP-N2F model has significant structure-learning ability on symbolic reasoning tasks.

2 Background: Review of the Tensor Product Representation

The TPR mechanism is a method to create a vector-space embedding of complex symbolic structure. The type of a symbol structure is defined by a set of structural positions or **roles**, such as the left-child-of-root position in a tree, or the second-argument position of a particular relation. In a particular instance of a structural type, each of these roles may be occupied by particular **filler**, which can be an atomic symbol or a substructure (e.g., the left sub-tree of a binary tree can serve as the filler of the role left-child-of-root). For now, we assume the fillers to be atomic symbols.

The TPR embedding of a symbolic structure is the sum of the embeddings of all its constituents, each constituent comprising a role together with its filler. The embedding of a constituent is constructed from the embedding of a role and the embedding of the filler of that role: these are joined together by the TPR **binding** operation: the tensor (or generalized outer) product $\otimes$.

Thus if a symbolic type is defined by the roles $\{r_i\}$, and in a particular instance of that type, $\mathtt{S}$, role r_i is bound by filler s_i, the TPR embedding of $\mathtt{S}$ is

$$\mathsf{T} = \sum_i \boldsymbol{s}_i \otimes \boldsymbol{r}_i = \sum_i \boldsymbol{s}_i \boldsymbol{r}_i^\top \tag{1}$$

where $\{\boldsymbol{s}_i\}$ are vector embeddings of the fillers and $\{\boldsymbol{r}_i\}$ are vector embeddings of the roles. Define the matrix of all n_R possible role vectors to be $\boldsymbol{R} \in \mathbb{R}^{d_\mathrm{R} \times n_\mathrm{R}}$, with column i, $[\boldsymbol{R}]_{:i} = \boldsymbol{r}_i \in \mathbb{R}^{d_\mathrm{R}}$, comprising the embedding of r_i. Similarly let $\boldsymbol{F} \in \mathbb{R}^{d_\mathrm{F} \times n_\mathrm{F}}$ be the matrix of all possible filler vectors. The TPR $\mathsf{T} \in \mathbb{R}^{d_\mathrm{F} \times d_\mathrm{R}}$.

Choosing the tensor product as the binding operation makes it possible to recover the filler of any role in a structure $\mathtt{S}$ given the TPR T of $\mathtt{S}$. This can be done with perfect precision if the embeddings of the roles are linearly independent. In that case the role matrix $\boldsymbol{R}$ is invertible: $\boldsymbol{U} = \boldsymbol{R}^{-1}$ exists such that $\boldsymbol{U}\boldsymbol{R} = \boldsymbol{I}$. Now define the **unbinding** vector for role r_j, $\boldsymbol{u}_j$, to be the j^{th} column of $\boldsymbol{U}^\top$: $U_{:j}$. Then, since $[\boldsymbol{I}]_{ji} = [\boldsymbol{U}\boldsymbol{R}]_{ji} = \boldsymbol{U}_{j:}\boldsymbol{R}_{:i} = [\boldsymbol{U}_{:j}^\top]^\top \boldsymbol{R}_{:i} = \boldsymbol{u}_j^\top \boldsymbol{r}_i = \boldsymbol{r}_i^\top \boldsymbol{u}_j$, we have $\boldsymbol{r}_i^\top \boldsymbol{u}_j = \delta_{ji}$. This means that, to recover the filler of r_j in the structure with TPR T, we can take the inner product with $\boldsymbol{U}_j$ (or equivalently, viewing T as a matrix, take the matrix-vector product):

$$\mathsf{T}\boldsymbol{u}_j = \left[\sum_i \boldsymbol{s}_i \boldsymbol{r}_i^\top\right] \boldsymbol{u}_j = \sum_i \boldsymbol{s}_i \delta_{ij} = \boldsymbol{s}_j \tag{2}$$

In the architecture proposed here, we will make use of both TPR-binding using the tensor product with role vectors and TRP-unbinding using the inner product with unbinding vectors.

3 TP-N2F Model

We proposed a TP-N2F model supported by a novel TP-N2F scheme over TPRs to map natural descriptions to formal expressions. In TP-N2F scheme, natural language is represented as a role-filler scheme of TPRs. Relational representations over structures are represented as a novel recursive role-filler scheme of TPRs proposed here. Figure 1 shows an overview diagram of TP-N2F model.

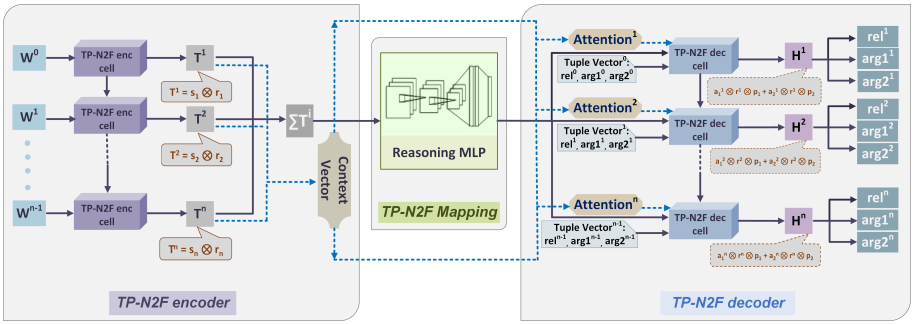

Figure 1: Overview diagram of TP-N2F.

As showed in Figure 1, while the natural language input is a sequence of words, the output is a sequence of multi-argument **relational tuples** such as $(rel\ arg1\ arg2)$, consisting of a relation rel with its two arguments. TP-N2F contains a "TP-N2F encoder", which encodes natural-language sequence via TPR-binding, and a "TP-N2F decoder", which decodes relational tuples via TPR-unbinding. In the following sections, we firstly introduce the details of TP-N2F scheme, and then we present how TP-N2F model uses the TP-N2F scheme and binding-unbinding operations of TPRs on the N2F task.

3.1 TP-N2F Scheme

In this section, We explain TP-N2F scheme, which defines the natural-language symbolic expressions and relational representations.

3.1.1 TP-N2F scheme for natural language

Instead of encoding each token of a sentence with a embedding vector, TPRs uses a role-filler scheme to represent the structural meaning of natural language. Given a sentence S with n word tokens $\{w_0, w_1, ..., w_n\}$, each word token w_i has a role vector $\boldsymbol{r}_i$ for grammar meaning and a filler vector $\boldsymbol{s}_i$ for semantic meaning. Then each word token is represented by the tensor product of the role vector and the filler vector as following:

$$\mathbf{T}_{w_i} = \boldsymbol{s}_i \otimes \boldsymbol{r}_i \tag{3}$$

The TPRs of a sentence S is the sum of TPRs embeddings of all word tokens $\{w_0, w_1, ..., w_n\}$ as $\sum_{i=0}^{n} \mathbf{T}_{w_i}$. TPRs of natural language has several advantages. Firstly, natural language TPR can be interpreted by exploring the distribution of tokens grouped by role and filler vectors. Secondly, TPRs avoids the Bag of Word (BoW) confusion. For example, in BoW embedding, the vector that encodes "Jay saw Kay" is the same as the one that encodes "Kay saw Jay" (8). However, in TPRs embedding, $\boldsymbol{s}_{Jay} \otimes \boldsymbol{r}_{Jay} + \boldsymbol{s}_{saw} \otimes \boldsymbol{r}_{saw} + \boldsymbol{s}_{Kay} \otimes \boldsymbol{r}_{Kay}$ is different from $\boldsymbol{s}_{Kay} \otimes \boldsymbol{r}_{Kay} + \boldsymbol{s}_{saw} \otimes \boldsymbol{r}_{saw} + \boldsymbol{s}_{Jay} \otimes \boldsymbol{r}_{Jay}$, because the filler and role vectors are changed based on context.

3.1.2 TP-N2F Scheme for Relational Representations

In this paper, we proposed a novel recursive role-filler scheme in vector-space based on TPRs to represent any symbolic relational tuples. Each relational tuples contains a relation token and multiple argument tokens.

Given a binary relation rel which needs two arguments, a relational tuples can be written as $(rel \quad arg1 \quad arg2)$ where $arg1, arg2$ indicate two arguments of relation rel. Suppose vector $\boldsymbol{r}, \boldsymbol{a}_1, \boldsymbol{a}_2$ represents the embedding vector for relation rel and arguments $arg1, arg2$, the TPR for the relational tuples $(rel \quad arg1 \quad arg2)$ is:

$$\mathbf{T}_F = \boldsymbol{a}_1 \otimes \boldsymbol{r} \otimes \boldsymbol{p}_1 + \boldsymbol{a}_2 \otimes \boldsymbol{r} \otimes \boldsymbol{p}_2 \tag{4}$$

$\boldsymbol{p}_1, \boldsymbol{p}_2$ are the positional vectors to indicate the positional information of each argument in the relation R. Each third order tensor product in the sum is regarded as a two-level recursive role-filler scheme.

Each positional vectors is the first-level role vector representing positional role and the tensor product between argument and relation is the first-level filler vector. The tensor product of argument and relation is another role-filler pair where the relation is relational role vector and argument is the filler vector.

Given the unbinding vector for positional roles p'_i and relational roles r'_i, each argument can be unbinded with two steps as shown in (5) and (6). This scheme can be extended to multi-arguments relational tuples and avoid the dimension exploding, like $\boldsymbol{r} \otimes \boldsymbol{a}_1 \otimes \boldsymbol{a}_2 \otimes \boldsymbol{a}_3$ in (10).

$$\boldsymbol{a}_i \otimes \boldsymbol{r} = [\boldsymbol{a}_1 \otimes \boldsymbol{r} \otimes \boldsymbol{p}_1 + \boldsymbol{a}_2 \otimes \boldsymbol{r} \otimes \boldsymbol{p}_2]\, \boldsymbol{p}'_i = \mathbf{T}_F \boldsymbol{p}'_i \tag{5}$$

$$\boldsymbol{a}_i = [\boldsymbol{a}_i \otimes \boldsymbol{r}]\, \boldsymbol{r}'_i \tag{6}$$

3.1.3 TP-N2F Scheme for Learning

To generate formal relational tuples from natural language, learning strategy for the mapping between the two structures is particularly important. We formalize the learning scheme as a learning mapping function $f_{mapping}(.)$, which generates structural information of formal language $I_F(.)$ from the structural information of natural language $I_N(.)$ like the (7) noted. In this paper, we use a MLP module as the TP-N2F mapping-learning-scheme $f_{mapping}(.)$ to learn the structural mapping. We will test various models to learn structural mapping functions in future works.

$$I_F(\mathbf{T}_F) = f_{mapping}(I_N(\mathbf{T}_S)) \tag{7}$$

3.2 TP-N2F Model for Natural- to- Formal- Language Generation

As shown in Figure 1, TP-N2F model is implemented with three steps: encoding, mapping, and decoding. Encoding step is implemented by TP-N2F natural language encoder (*TP-N2F encoder*), which takes the sequence of word tokens as inputs, and encodes them to the TP-N2F scheme for natural language via TPRs-binding. Mapping step is implemented by a MLP, called reasoning module, which takes the encoding output of TP-N2F encoder as input. The reasoning module learns to map the natural language structure encoding to a vector containing the structure information of relational tuples as described in section 3.1.3. Decoding step is implemented by TP-N2F relational tuples decoder (*TP-N2F decoder*), which takes the output from the reasoning MLP mapping and decodes the targeted sequence of relational tuples via TPRs-unbinding. An attention mechanism is utilized for TP-N2F decoder. The implementation details of TP-N2F encoder, TP-N2F decoder and learning strategy are introduced in turn.

3.2.1 TP-N2F Natural Language Encoder

TP-N2F encoder uses the scheme in section 3.1.1 to encode each word token w_t by selecting one filler from $nFillers$ fillers and one role from $nRoles$ roles. The fillers and roles are embedded as vectors and these embedding vectors are learned by two LSTMs, Filler-LSTM and Role-LSTM.

At each timestep t, Filler-LSTM and Role-LSTM takes a word token $\boldsymbol{w}^t$ as input. The hidden state of Filler-LSTM $\boldsymbol{h}f^t$ is used to compute the softmax scores on $nFillers$ filler slots and a filler $\boldsymbol{f}^t$ vector is learned from the softmax scores. Similarly, a role vector is learned from the hidden state of Role-LSTM $\boldsymbol{h}r^t$. f_f and f_r indicate the process to generate $\boldsymbol{f}^t$ and $\boldsymbol{r}^t$ from hidden states of two LSTMs. The token w^t is encoded as $\mathbf{T}^t$, the tensor product of $\boldsymbol{f}^t$ and $\boldsymbol{r}^t$. $\mathbf{T}^t$ replace the hidden vector in each LSTM and passed to the next timestep together with context vector C^t. The details are described in (8-9). After encoding the whole sequence, TP-N2F encoder outputs the sum of all tensor products $\sum \mathbf{T}^t$ to the next module. Figure 2 shows the detailed implementation diagram of TP-N2F encoder.

$$\boldsymbol{h}_f^t = f_{filler-lstm}(w^t, \mathbf{T}^t, \boldsymbol{c}_s^t) \qquad \boldsymbol{h}_r^t = f_{role-lstm}(w^t, \mathbf{T}^t, \boldsymbol{c}_r^t) \tag{8}$$

$$\mathbf{T}^t = \boldsymbol{f}^t \otimes \boldsymbol{r}^t = f_f(\boldsymbol{h}_f^t) \otimes f_r(\boldsymbol{h}_r^t) \tag{9}$$

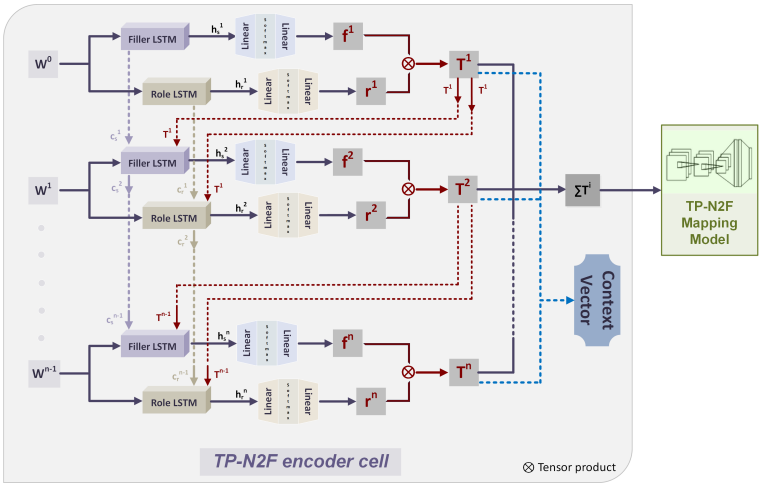

Figure 2: Implementation of TP-N2F encoder.

3.2.2 TP-N2F Relational Tuples Decoder

TP-N2F decoder takes the output from reasoning MLP as initial hidden state and decode a sequence of relational tuples. TP-N2F decoder contains a LSTM called Tuple-LSTM and an unbinding module.

At each timestep t, the hidden state H^t of Tuple-LSTM is assumed as a TPR of a relational tuple with m arguments $H^t = \sum_{i=1}^{m} a_i^t \otimes r^t \otimes p_i$ using the scheme in section 3.1.2. Then, the unbinding module decodes the relational tuple from hidden state H^t using two steps TPR-unbinding as (5) and (6). The unbinding positional vectors p_i^t are learned during training and shared in all timesteps. After the first unbinding step, we get B_i^t as outputs. The unbinding relational vector is computed from the sum of all B_i^t using a linear function. Finally, the arguments and relation of the relational tuple at each timestep are classified from the unbinded argument vectors and generated unbinding relational vector. Formulas (10-12) and Figure 3 shows a TP-N2F decoder diagram to decode binary relational tuples.

$$\mathbf{H}^t = f_{tuple-lstm}(rela^t, arg1^t, arg2^t, \mathbf{H}^t, c^t) \tag{10}$$

$$\overset{t}{1} = \mathbf{H}^t \boldsymbol{p}_1' \qquad \overset{t}{2} = \mathbf{H}^t \boldsymbol{p}_2' \tag{11}$$

$$\boldsymbol{r}'^t = f_{linear}(\overset{t}{1} + \overset{t}{2}) \qquad \boldsymbol{a}_1^t = \overset{t}{1}\, \boldsymbol{r}'^t \qquad \boldsymbol{a}_2^t = \overset{t}{2}\, \boldsymbol{r}'^t \tag{12}$$

3.3 The Learning Strategy of TP-N2F Model

TP-N2F is trained using back-propagation with Adam optimizer (11) and teacher-forcing strategy. At each timestep, the ground truth relational tuple is provided as the input for next timestep. As TP-N2F decoder decodes a relational tuple at each timestep, the relation token is selected only from relation vocabulary and argument tokens are selected from argument vocabulary. The loss is obtained by summing the cross-entropy loss F_{loss} between true labels L and predicated tokens of relations and arguments as shown in (13).

$$loss = \sum_{i=0}^{n} [F_{loss}(rela^i, L_{rela^i})] + \sum_{j=0}^{m} \sum_{i=0}^{n} [F_{loss}(arg_j^i, L_{arg_j^i})] \tag{13}$$

4 Experiments

Our TP-N2F model is tested on two instance of N2F tasks: 1) operation sequence generation to solve math problems; 2) Lisp-program generation. In both tasks, TP-N2F achieves the state-of-the-art performance. The details of each task are introduced below.

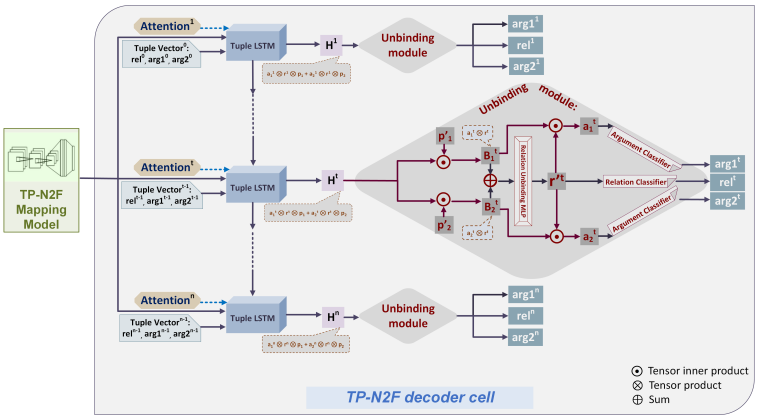

Figure 3: Implementation of TP-N2F decoder.

4.1 Generating operation sequence to solve math problems

Generating human-like operation sequence to solve math problems is challenging. Given a natural-language math problem, we need to generate a sequence of operations (operators and corresponding arguments) from a set of operators and arguments to solve the problems. Each operation is regarded as a relational tuple by viewing operator as relation. We test TP-N2F for this task on MathQA dataset for this task and get the state of the art performance.

4.1.1 MathQA dataset

MathQA dataset (12) consists of about 37k math word problems, corresponding lists of multiple-choice options and aligned operation sequences. All data samples are randomly split in (80/12/8)% training/dev/testing problems. In this task, TP-N2F is deployed to generate the operation sequence, which can be executed to solve math problems.

We use the same execution script from (12) to execute the generated operation sequence and compute the multi-choice accuracy for each problem. As there are about 30% noise data (execution script cannot choose the right option on ground truth), both the execution accuracy and operation sequence accuracy (generated operation sequence matches ground truth exactly) to evaluate our model.

4.1.2 Results

TP-N2F is compared with seq2prog proposed in (12), which is a LSTM Seq2Seq model with attention model and achieves state-of-the-art performance. Our model outperforms both the original setting seq2prog, indicated as SEQ2PROG(org), and the best duplicated seq2prog after a random hyper-parameter search, indicated as SEQ2PROG(best). Ablation experiments on TP2LSTM (replace the TP-N2F decoder as a LSTM with hidden size 100) and LSTM2TP (replace the TP-N2F encoder as a LSTM with hidden size 100) show that both TP-N2F encoder and TP-N2F decoder are important for this task. Table 1 shows the performance result.

Table 1: Results on MathQA dataset testing set

MODEL	Operation Accuracy(%)	Execution Accuracy(%)
SEQ2PROG-org (12)	59.4	51.9
SEQ2PROG-best (12)	66.97	54.0
TP2LSTM (ours)	68.84	54.61
LSTM2TP (ours)	68.21	54.61
TP-N2F (ours)	**71.89**	**55.95**

Table 2: Results of AlgoLisp dataset

	Full Testing Set			Cleaned Testing Set		
MODEL (%)	**Acc**	**50p-Acc**	**M-Acc**	**Acc**	**50p-Acc**	**M-Acc**
LSTM2LSTM+atten	67.54%	70.89%	75.12%	76.83%	78.86%	75.42%
TP2LSTM (ours)	72.28%	77.62%	79.92%	77.67%	80.51%	76.75%
LSTM2TPR (ours)	75.31%	79.26%	83.05%	84.44%	86.13%	83.43%
SAPSpre-VH-Att (256) (14)	83.80%	87.45%		92.98%	94.15%	
TP-N2F (ours)	**84.02%**	**88.01%**	**93.06%**	**93.48%**	**94.64%**	**92.78%**

4.2 Generating Lisp program tree

Generating Lisp program needs the awareness of structural information because Lisp codes can be regarded as tree structures. Given a natural-language query, we need to generate codes containing function calls and their parameters. Each function call is a relational tuple, which has function as the relation and parameters as arguments. We test TP-N2F on AlgoLisp dataset for this task and get state-of-the-art-performance.

4.2.1 AlgoLisp dataset

AlgoLisp dataset (13) is a program synthesis dataset, which has about 79k/9k/10k training/dev/testing samples. Each sample contains a problem description, corresponding Lisp program tree and 10 input-output testing pairs. We parse the program tree to a sequence of commands from leave to root and use symbol $\#_i$ to indicate the i^{th} command the model generated before. AlgoLisp provides an execution script to run the generated program and has three evaluate matrices: accuracy of passing all testing cases (Acc), accuracy of passing 50% testing cases (50p-Acc), and accuracy of generating exact matched program (M-Acc). As AlgoLisp has about 10% noise data (execution script fails passing all test cases on ground truth), We report the results both on full testing set and cleaned testing set (remove all noise testing samples).

4.2.2 Results

TP-N2F is compared with LSTM Seq2Seq with attention model and Seq2Seq model with a pre-trained tree-decoder from Tree2Tree autoencoder (SAPS) in (14). TP-N2F outperforms all other models and achieve state-of-the-art performance on both full testing set and cleaned testing set. Ablation experiments on TP2LSTM and LSTM2TP show that TP-N2F decoder is more helpful than TP-N2F encoder in this task. Table 2 shows the results.

5 Related Work

TPR is a promising scheme to encode symbolic structural information and models symbolic reasoning in vector-space. TPR-binding has been used for encoding grammatical structure of natural language on question-answering tasks and encoding symbolic structure information (7). TPRs is also used to unbind natural language captions from images and present grammatical structures of natural language (8; 9). Some researchers use TPRs for modeling deductive reasoning process both on rule-based model and deep learning models in vector-space (15; 16; 10). However, all of these works do not take the advantages of combining TPR-binding and TPR-unbinding in deep learning models to learn structure representation mappings explicitly like our model.

Many researchers have explored generating formal-language from natural language descriptions in many works. For example, operations sequence generation from math problems is formalized as a machine translation task on word-token level and a modified version of LSTM Seq2Seq is deployed on this task (12). Program generation is regarded as decoding tree structure from natural language and tree decoder is utilized to generate program tree (13; 14). Although people pay increasing attention on N2F task, most of the proposed models either do not encode structural information explicitly or only fit specific tasks. In this paper, our proposed TP-N2F scheme and model can be applied to multiple tasks.

6 Conclusion and future work

In this work, we proposed a new scheme for neural-symbolic relational representations and a new architecture, TP-N2F, on formal-language generation from natural-language. To our knowledge, TP-N2F is the first model that combines TPRs-binding and TPRs-unbinding together in the encoder-decoder fashion. TP-N2F achieves the state-of-the-art on two instances of N2F task and shows strong structure learning ability. The results show that both TP-N2F encoder and TP-N2F decoder are important for generating qualitative structures. In the future, we plan to combine the large-scale deep learning models such as BERT with TP-N2F for various tasks. We also want to take the advantage of TPRs in deep learning models for more complicated knowledge reasoning and representation tasks.

References

[1] S. Goldin-Meadow and D. Gentner, *Language in mind: Advances in the study of language and thought*. MIT Press, 2003.

[2] K. Forbus, C. Liang, and I. Rabkina, "Representation and computation in cognitive models," in *Top Cognitive System*, 2017.

[3] M. Crouse, C. McFate, and K. D. Forbus, "Learning from unannotated qa pairs to analogically disanbiguate and answer questions," in *Thirty-Second AAAI Conference*, 2018.

[4] K. Chen and K. D. Forbus, "Action recognition from skeleton data via analogical generalization over qualitative representations," in *Thirty-Second AAAI Conference*, 2018.

[5] K. Chen, I. Rabkina, M. D. McLure, and K. D. Forbus, "Human-like sketch object recognition via analogical learning," in *Thirty-Third AAAI Conference*, vol. 33, 2019, pp. 1336–1343.

[6] P. Smolensky, "Tensor product variable binding and the representation of symbolic structures in connectionist networks," in *Artificial Intelligence*, vol. 46, 1990, pp. 159–216.

[7] H. Palangi, P. Smolensky, X. He, and L. Deng, "Question-answering with grammatically-interpretable representations," in *AAAI*, 2018.

[8] Q. Huang, P. Smolensky, X. He, O. Wu, and L. Deng, "Tensor product generation networks for deep nlp modeling," in *NAACL*, 2018.

[9] Q. Huang, L. Deng, D. Wu, c. Liu, and X. He, "Attentive tensor product learning," in *Thirty-Third AAAI Conference*, vol. 33, 2019.

[10] I. Schlag and J. Schmidhuber, "Learning to reason with third order tensor products," in *Neural Information Processing Systems*, 2018.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2017.

[12] A. Amini, S. Gabriel, P. Lin, R. K. Kedziorski, Y. Choi, and H. Hajishirzi, "Mathqa: Towards interpretable math word problem solving with operation-based formalisms," in *NACCL*, 2019.

[13] I. Polosukhin and A. Skidanov, "Neural program search: Solving programming tasks from description and examples," in *ICLR workshop*, 2018.

[14] J. Bednarek, K. Piaskowski, and K. Krawiec, "Ain't nobody got time for coding: Structure-aware program synthesis from natural language," in *arXiv.org*, 2019.

[15] M. Lee, X. He, W.-t. Yih, J. Gao, L. Deng, and P. Smolensky, "Reasoning in vector space: An exploratory study of question answering," in *ICLR*, 2016.

[16] P. Smolensky, M. Lee, X. He, W.-t. Yih, J. Gao, and L. Deng, "Basic reasoning with tensor product representations," *arXiv preprint arXiv:1601.02745*, 2016.