Grammatically-Interpretable Learned Representations in Deep NLP Models

Hamid Palangi[*]
Microsoft Research AI
Redmond, WA
hpalangi@microsoft.com

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

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

Xiaodong He
Microsoft Research AI
Redmond, WA
xiaohe@microsoft.com

Li Deng
Microsoft Research AI
Redmond, WA
l.deng@ieee.org

Abstract

We introduce two architectures, the Tensor Product Recurrent Network (TPRN) and the Tensor Product Generation Network (TPGN). In the application of TPRN, internal representations — learned by end-to-end optimization in a deep neural network performing a textual QA task — are interpretable using basic concepts from linguistic theory. This interpretability is achieved without paying a performance penalty. In another application, image-to-text generation or image captioning, TPGN gives better results than the state-of-the-art long short-term memory (LSTM) based approaches. Learned internal representations in the TPGN can also be interpreted as containing grammatical-role information.

1 Introduction

The difficulty of explaining the operation of deep neural networks begins with the difficulty of interpreting the internal representations learned by these networks. This difficulty could in principle be reduced if deep neural networks were to incorporate internal representations that are directly interpretable as discrete structures; the categories and relations of these representations might then be understandable conceptually.

The work reported here shows how approximately discrete, structured distributed representations can be embedded within deep networks, their categories and structuring relations being learned end-to-end through performance of a task. The tasks we address are question-answering for the SQuAD dataset [20] and caption-generation for the MS-COCO data [5].

The proposed capacity for distributed representation of structure is provided by Tensor Product Representations, TPRs, in which a discrete symbol structure is encoded as a vector systematically built—through vector addition and the tensor product—from vectors encoding symbols and vectors encoding the roles each symbol plays in the structure as a whole [22, 24, 25].

The first model presented here is built from the BIDAF model proposed in [21] for question answering. We replace bidirectional RNNs built from LSTM units [11] with ones built from TPR units; the architecture is called the *Tensor Product Recurrent Network*, TPRN [16]. TPRN learns the vector embeddings of the symbols and roles, and learns which abstract symbols to deploy in which abstract roles to represent each of the words in the text-passage and query inputs.

[*]This work was carried out while PS was on leave from Johns Hopkins University. LD is currently at Citadel.

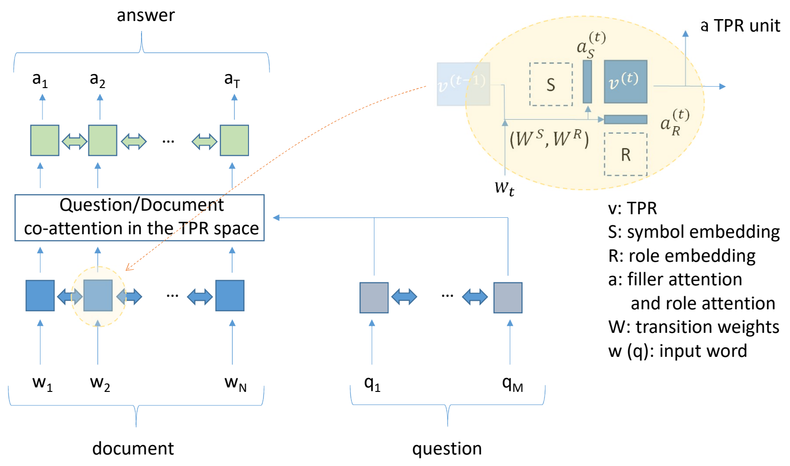


Figure 1: Block diagram of the proposed TPRN question-answering model.

A second model we discuss uses the structure of TPR computation to inform the design of an architecture for natural language generation, the *Tensor Product Generation Network* TPGN, which we apply to image captioning. Again it turns out that the structural roles learned by end-to-end deep learning can be interpreted grammatically.

The models presented here show how learning to perform a typical natural language task can lead a deep learning system to create representations that are interpretable as encoding abstract grammatical concepts without ever being exposed to data labelled with anything like grammatical structure.

The remainder of the paper is structured as follows. Section 2 introduces TPR and details how it is used in the general TPRN architecture we propose here. Experimental results applying TPRN to question-answering with SQuAD are presented in Section 2.1. The focus of this work is addressed in Section 2.2 which presents interpretations of the representations learned by TPRN. Section 3 presents the TPGN architecture and applies it to caption generation with the COCO dataset; experimental results are presented in Section 3.2. Section 4 briefly discusses related work and Section 5 concludes.

2 TPRN: The Tensor Product Recurrent Network

In the SQuAD dataset, a text passage and a question are presented as input, and the model's output identifies a stretch within the passage that contains the answer to the question. The proposed TPRN architecture is built in TensorFlow [1] on the Bidirectional Attention Flow (BIDAF) model proposed in [21]. BIDAF is constructed from 6 layers: a character-embedding layer using CNNs, a word-embedding layer using GloVe vectors [18], a phrase-embedding layer using bidirectional LSTMs for embedding words in sentential context [15], an attention-flow layer using a special attention mechanism, a modeling layer using LSTMs, and an output layer that generates pointers to the start and end of an answer in the paragraph. (See Fig. 1 of [21].)

The first version of TPRN replaces the LSTM cells forming the bidirectional RNN in the phrase embedding layer with recurrent TPR cells, described next: see Fig. 1.

This TPRN model enables the phrase-embedding layer of the model to decide, for each word, how to encode that word by selecting among n_S symbols, each of which it can choose to deploy in any of n_R slots in an abstract latent structure. The symbols and slots have no meaning prior to training. We hypothesized that the symbol selected by the trained model for encoding a given input word will be interpretable in terms of the lexical-semantic content of the word (e.g., *Australia* refers to a place) while the slots will be interpretable as grammatical roles such as subject/agent, object/patient, question-restrictor phrase. In Section 2.2, we will test this hypothesis; we will henceforth refer to "roles" rather than "slots". In other words, our hypothesis was that the particular word tokens for which a given symbol was selected would form a lexical-semantically-related class, and the particular word tokens for which a given role was selected would form a grammatically-related class.

To function within the network, the symbols and roles must each be embedded as vectors; assume that we use vectors of dimension d_S and d_R for symbols and roles respectively. These embedding vectors are designed by the network, i.e., they are learned during training. The network's parameters, including these embeddings, are driven by back-propagation to minimize an objective function relevant to the model's question-answering task. The objective function includes a standard cross-entropy error measure, but also *quantization*, a kind of regularization function biasing the model towards parameters which yield decisions that select, for each word, a single symbol in a single role: the selection of symbols and roles is soft-selection, and we will say that the model's encoding assigns, to the t^{th} word $w^{(t)}$, a *symbol-attention vector* $\mathbf{a}_\mathrm{S}{}^{(t)}$ and a *role-attention vector* $\mathbf{a}_\mathrm{R}{}^{(t)}$.

The quantization term in the objective function pushes towards attention vectors that are 1-hot. We do not impose this as a hard constraint because our fundamental hypothesis is that by developing *approximately* discrete representations, the model can benefit from the advantages of discrete combinatorial representations for natural language, without suffering their disadvantage of rigidity. We note that while the attention vectors are approximately 1-hot, the actual representations deployed are attention-weighted sums of fully distributed vectors arising from distributed encodings of the symbols and distributed embeddings of the roles.

In the encoding for $w^{(t)}$, the vector $\mathbf{s}^{(t)}$ encoding the symbol is the attention-weighted sum of the n_S possible symbols: $\mathbf{s}^{(t)} = \sum_{j=1}^{n_S} [\mathbf{a}_\mathrm{S}{}^{(t)}]_j \mathbf{s}_j = \mathbf{S}\mathbf{a}_\mathrm{S}{}^{(t)}$ where $\mathbf{s}_j$ is the embedding of the j^{th} symbol in $\mathbb{R}^{d_S}$, which is the j^{th} column of the *symbol matrix* $\mathbf{S}$. Similarly, the vector encoding the role assigned to $w^{(t)}$ is $\mathbf{r}^{(t)} = \sum_{k=1}^{n_R} [\mathbf{a}_\mathrm{R}{}^{(t)}]_k \mathbf{r}_k = \mathbf{R}\mathbf{a}_\mathrm{R}{}^{(t)}$, with $\mathbf{r}_k$ the embedding of the k^{th} symbol in $\mathbb{R}^{d_R}$ and the k^{th} column of the *role matrix* $\mathbf{R}$.

The activation vector $\mathbf{v}^{(t)}$ that encodes a single word $w^{(t)}$ combines the word's symbol-embedding vector, $\mathbf{s}^{(t)}$, and its role-embedding vector, $\mathbf{r}^{(t)}$, via the outer or tensor product: $\mathbf{v}^{(t)} = \mathbf{a}_\mathrm{S}{}^{(t)}\mathbf{a}_\mathrm{R}{}^{(t)\top} = \mathbf{a}_\mathrm{S}{}^{(t)} \otimes \mathbf{a}_\mathrm{R}{}^{(t)}$. We say that $\mathbf{v}^{(t)}$ is the *tensor product representation (TPR) of the binding of symbol* $\mathbf{s}^{(t)}$ *to the role* $\mathbf{r}^{(t)}$. A convenient expression for $\mathbf{v}^{(t)}$ is:

$$\mathbf{v}^{(t)} \equiv \mathbf{s}^{(t)}(\mathbf{r}^{(t)})^\top = \left(\mathbf{S}\mathbf{a}_\mathrm{S}{}^{(t)}\right)\left(\mathbf{R}\mathbf{a}_\mathrm{R}{}^{(t)}\right)^\top = \mathbf{S}\left(\mathbf{a}_\mathrm{S}{}^{(t)}\mathbf{a}_\mathrm{R}{}^{(t)\top}\right)\mathbf{R}^\top = \mathbf{S}\mathbf{B}^{(t)}\mathbf{R}^\top \tag{1}$$

The matrix $\mathbf{B}^{(t)} \equiv \mathbf{a}_\mathrm{S}{}^{(t)}\mathbf{a}_\mathrm{R}{}^{(t)\top}$ is the *binding matrix* for word $w^{(t)}$, which encodes the (soft) selection of symbol and role for $w^{(t)}$. This matrix has dimension $n_S \times n_R$; the actual representation sent to deeper layers, $\mathbf{v}^{(t)}$, has $d_S \times d_R$ embedding dimensions. (E.g., in one particular model discussed below, the dimensions of $\mathbf{B}$ and $\mathbf{v}^{(t)}$ are respectively 100×20 and 10×10.)

$\mathbf{a}_\mathrm{S}{}^{(t)}$ and $\mathbf{a}_\mathrm{R}{}^{(t)}$ in (1) are computed by:

$$\mathbf{a}_\mathrm{S}{}^{(t)} = f(\mathbf{W}_{in}^S\mathbf{w}^{(t)} + \mathbf{W}_{rec}^S vec(\mathbf{v}^{(t-1)}) + \mathbf{b}^S) \tag{2}$$

$$\mathbf{a}_\mathrm{R}{}^{(t)} = f(\mathbf{W}_{in}^R\mathbf{w}^{(t)} + \mathbf{W}_{rec}^R vec(\mathbf{v}^{(t-1)}) + \mathbf{b}^R) \tag{3}$$

where $vec(.)$ is the vectorization operation, $f(.)$ is the logistic sigmoid function, $\mathbf{w}^{(t)}$ is the t^{th} word and $\mathbf{b}$ is a bias vector. Equation (1) is depicted graphically in the 'TPR unit' insert in Fig. 1. During $I \to O$ inference, in the forward-directed RNN, the representation $\mathbf{v}^{(\mathbf{t-1})}$ of the previous word is used to compute the attention vectors $\mathbf{a}_\mathrm{S}{}^{(t)}$, $\mathbf{a}_\mathrm{R}{}^{(t)}$ which in turn are used to compute the representation $\mathbf{v}^{(\mathbf{t})}$ of the current word. (The same equations, with the same transition weights and biases, apply to the words in both the passage and the query.) $\mathbf{v}^{(t)}$ is initialized to zero.

Because each word is represented (approximately) as the TPR of a single symbol/role binding, we can interpret the internal representations of TPRN 's phrase-embedding layer once we can interpret the symbols and roles it has invented. Such interpretation is carried out in Section 2.2.

The interest in TPR lies not only in its interpretability, but also in its power. The present TPRN model incorporates TPR to only a modest degree, but it is a proof-of-concept system that paves the way for future models that can import the power of general symbol-structure processing, proven to be within the scope of full-blown TPR architectures [24, 23]. TPRN is designed to scale up to such architectures; design decisions such as factoring the encoding as $\mathbf{v}^{(t)} = \mathbf{a}_\mathrm{S}{}^{(t)}\mathbf{a}_\mathrm{R}{}^{(t)\top} = \mathbf{a}_\mathrm{S}{}^{(t)} \otimes \mathbf{a}_\mathrm{R}{}^{(t)}$ are far from arbitrary: they derive directly from the general TPR architecture.

As the name TPRN suggests, the novel representational capacity built into TPRN is an RNN built of TPR units: a forward- and a backward-directed RNN in each of which the word $w^{(t)}$ generates an encoding which is a TPR: $\mathbf{v}^{(t)} = \mathbf{S}\mathbf{B}^{(t)}\mathbf{R}^{\top}$; the binding matrix $\mathbf{B}^{(t)}$ varies across words, but a single symbol matrix $\mathbf{S}$ and single role matrix $\mathbf{R}$ apply for all words $\{w^{(t)}\}$. Both the $\mathbf{S}$ and $\mathbf{R}$ matrices are learned during training. In addition, a standard LSTM gating mechanism operates over $\mathbf{v}^{(t)}$.

It remains only to specify the quantization function $\mathcal{Q}$ (4) which is added (with weight c_Q) to the cross-entropy to form the training objective for TPRN : $\mathcal{Q}$ generates a bias favoring attention vectors $\mathbf{a}_{\mathrm{S}}{}^{(t)}$ and $\mathbf{a}_{\mathrm{R}}{}^{(t)}$ that are 1-hot.

$$\mathcal{Q} = \mathcal{Q}_a(\mathbf{a}_{\mathrm{S}}{}^{(t)}) + \mathcal{Q}_a(\mathbf{a}_{\mathrm{R}}{}^{(t)}); \mathcal{Q}_a(\mathbf{a}) = \Sigma_i (a_i)^2 (1-a_i)^2 + \left(\Sigma_i (a_i)^2 - 1\right)^2 \tag{4}$$

The first term of $\mathcal{Q}_a$ is minimized when each component of $\mathbf{a}$ satisfies $a_i \equiv [\mathbf{a}]_i \in \{0,1\}$, and the second term is minimized when $\|\mathbf{a}\|_2^2 = 1$; their sum is minimized when $\mathbf{a}$ is 1-hot [3].

2.1 Experiments

For the details of the experiments applying the TPRN model to the question-answering task of the Stanford's SQuAD dataset [20], see [16]. The results of primary interest are the interpretations of the learned representations, discussed in Section 2.2.

Performance results of our model compared to the strong BiDAF model proposed in [21] are presented in Table 1. We compared the performance of single models. For the BiDAF baseline, we ran the code published in [21] with the advised hyperparameters. We report results for three versions of the TPRN model: TPRN$_1$ uses TPRN units only in the phrase-embedding layer, while TPRN$_{\mathrm{all}}$ uses TPRN units to replace all the LSTMs in BiDAF. In the "deep TPR" model, dTPRN$_{\mathrm{all}}$, all LSTMs in BiDAF are replaced by gated recurrent networks in which the cells take the form shown in Table 2. In place of the outer product, at each time s the symbol vector f^s and (fixed) role vector r are inputs to a learned linear transformation E which binds them together to form the binding vector b^s, of dimension d_b; b^s is the output of the cell. As in [26], the combining transformation E should be invertible, enabling the (learned) unbinding linear transformation D: with input $[b^s; r]$, D should output f^s, which is the target vector in an L_2 penalty in the loss function during learning. (A sigmoid non-linearity follows matrix multiplication by both E and D.)

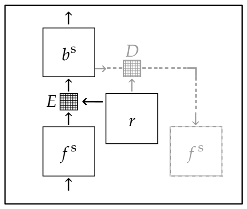
In the TPRN$_1$ model reported, we set the number of symbols (n_S) and roles (r_R) to 600 and 100 respectively and the embedding size of symbols (d_S) and roles (d_R) to 15 and 10. In TPRN$_{\mathrm{all}}$, n_S=100, n_R=20, d_S=10=d_R. For dTPR$_{all}$: n_S=200 n_R=50 d_b=60, d_S=12, $d_R = 4$.

From Table 1 we observe that TPRN$_1$ outperforms BiDAF by 1 point on the validation set and slightly underperforms BiDAF on the test set. The models with all LSTMs replaced by TPRNs achieve similar, but slightly lower, performance. Overall, TPRN gives results comparable to those of the state-of-the-art BiDAF model. Moreover, as we will now see, our model offers considerable interpretability thanks to the structure built into TPRs.

Table 1: Performance on development and test sets

Single Model	EM(dev)	F1(dev)	EM(test)	F1(test)
BiDAF [21]	62.8	73.5	67.1	76.8
TPRN$_1$	63.8	74.4	66.6	76.3
TPRN$_{\mathrm{all}}$	61.2	72.1	64.3	74.5
dTPRN$_{\mathrm{all}}$	63.2	73.4	65.9	75.8

Table 2: deep TPR cell



2.2 Experimental interpretations of learned TPRs

Here we consider interpretation of the TPR roles $\mathbf{a}_{\mathrm{R}}{}^{(t)}$ assigned to the words $w^{(t)}$ of the query input (denoted $q^{(t)}$ in Fig. 1) in the forward-directed TPR-RNN of a version of TPRN$_1$ in which $n_S = 100, n_R = 20; d_S = 10 = d_R$. We consider which word tokens $w^{(t)}$ are 'assigned to' (or 'select') a particular role k, meaning that, for an appropriate threshold θ_k, $[\hat{\mathbf{a}}_{\mathbf{R}}{}^{(t)}]_k > \theta_k$ where $\hat{\mathbf{a}}_{\mathbf{R}}{}^{(t)}$ is the L_2-normalized role-attention vector.

A grammatical category—Part of Speech: Determiner $\sim$ Role #9. The network assigns to role #9 these words: a significant proportion of the tokens of: *the* (76%), *an* (52%), *a* (46%), *its* (36%) and a few tokens of *of* (8%) and *Century* (3%). The dominant words assigned to role #9 (*the, an, a, its*) are all *determiners*. Although not a determiner, *of* is also an important function word; the 3% of the tokens of *Century* that activate role #9 can be put aside. Quantitatively, $p(w \text{ is a determiner}|w \text{ activates role } \#9 \text{ to } > 0.65) = 0.96$.

A semantic category: Predicate (verbs and adjectives) $\sim$ Role #17. The words assigned to role #17 are overwhelmingly predicates, a semantic category corresponding to the syntactic categories of verbs and adjectives [e.g., under semantic interpretation, *J runs $\rightarrow$ runs(J); J is tall $\rightarrow$ tall(J)*] . While the English word orders of these two types of predication are often opposite (*the girl runs* vs. *the tall girl*), the model represents them as both filling the same role, which can be interpreted as semantic rather than syntactic. Quantitatively, $p(w \text{ is a verb or adjective} | w \text{ selects role } \#17) = 0.82$.

A grammatical phrase-type: *wh*-operator restrictor 'phrase' $\sim$ Role #1. Role #1 is assigned to sequences of words including *how many teams, what kind of buildings, what honorary title*. We interpret these as approximations to a *wh-restrictor phrase*: a *wh*-word together with a property that must hold of a valid answer—crucial information for question-answering. In practice, these 'phrases' span from a *wh*-word to approximately the first following content word. Other examples are: *what was the American, which logo was, what famous event in history*.

CORRECTING A PART-OF-SPEECH (POS) TAGGER'S LABELING USING THE LEARNED ROLES

When *Doctor Who* is not a name: Role #7. The TV character Doctor Who (*DW*) is named many times in the SQuAD query corpus. Now in ... *DW travels* ... , the phrase *DW* is a proper noun ('NNP'), with unique referent, but in ... *does the first DW see* ..., the phrase *DW* must be a common noun ('NN'), with open reference. In such cases the Stanford tagger [13] misclassifies *Doctor* as an NNP in 9 of 18 occurrences. In ... *the first DW serial* ..., *first* modifies *serial* and *DW* is a proper noun. The tagger misparses this as an NN in 37 of 167 cases. Turning to the model, we can interpret it as distinguishing the NN vs. NNP parses of *DW* via role #7, which it assigns for the NN, but not the NNP, case. Of the Stanford tagger's 9 errors on NNs and 37 errors on NNPs, the model misassigns role #7 only once for each error type. The model makes 7 errors total while the tagger makes 46.

When *Who* is a name: Role #1. In *Doctor Who travelled*, the word *Who* should not be parsed as a question word ('WP'), but as part of a proper noun (NNP). The Stanford tagger makes this error in every one of the 167 occurrences of *Who* within the NNP *Doctor Who*. The TPRN model, however, usually avoids this error. Recalling that role #1 marks the *wh*-restrictor 'phrase', we note that in 81% of these NNP-*Who* cases, the model does not assign role #1 to *Who* (in the remaining cases, it does assign role #1 as it includes *Who* within its *wh*-restrictor 'phrase', generated by a distinct genuine *wh*-word preceding *Who*). In the 30 instances of *Who* as a genuine question word in a sentence containing *DW*, the model correctly assigns role #1 to the question word every time. (The model correctly selects role #1 for non-initial *who* in many cases.)

For further examples of grammatical interpretations of roles and corrections of the Stanford tagger, for interpretations of the symbols as lexical-semantic meanings (e.g., professions, geopolitical entities) and for explanation of model errors resulting from mis-representation of the question, see [16].

3 TPGN: Tensor Product Generation Networks

3.1 The model

In the work we consider next, we propose an approach to network architecture design we call the *TPR-capable method*. The architecture we use (see Fig. 2) is designed so that TPRs could, in theory, be used within the architecture to perform the target task — here, generating a caption one word at a time. Unlike TPRN, the learned representations are not constrained to be TPRs, but the architecture treats them as it would treat TPRs.

In Fig. 2, our proposed Tensor Product Generation Network (TPGN) system is denoted $\mathcal{N}$. The input to $\mathcal{N}$ is an image feature vector $\mathbf{v}$ and the output of $\mathcal{N}$ is a caption. The image feature vector $\mathbf{v}$ is extracted from a given image by a pre-trained CNN. The first part of our system $\mathcal{N}$ is a *sentence-encoding subnetwork* $\mathcal{S}$ which maps $\mathbf{v}$ to a representation S which will drive the entire caption-generation process; S contains all the image-specific information for producing the caption. (We will call a caption a "sentence" even though it may in fact be just a noun phrase.)

If S were a TPR of the caption itself, it would be a matrix (or 2-index tensor) $\mathbf{S}$ which is a sum of matrices, each of which encodes the binding of one word to its role in the sentence constituting the caption (see the $\mathbf{S}$ bubble in Fig. 2). To serially read out the words encoded in $\mathbf{S}$, in iteration 1 we would *unbind* the first word from $\mathbf{S}$, then in iteration 2 the second, and so on. As each word is generated, $\mathbf{S}$ could update itself, for example, by subtracting out the contribution made to it by the word just generated; $\mathbf{S}_t$ denotes the value of $\mathbf{S}$ when word w_t is generated. At time step t we would unbind the role r_t occupied by word w_t of the caption. So the second part of our system $\mathcal{N}$ — the *unbinding subnetwork* $\mathcal{U}$ — would generate, at iteration t, the *unbinding vector* $\mathbf{u}_t$, which is the dual of the vector $\mathbf{r}_t$ embedding the role r_t. Once $\mathcal{U}$ produces the unbinding vector $\mathbf{u}_t$, this vector would then be applied to $\mathbf{S}$ to extract the symbol $\mathbf{f_t}$ that occupies word t's role; the symbol represented by $\mathbf{f_t}$ would then be decoded into word w_t by the third part of $\mathcal{N}$, i.e., the *lexical decoding subnetwork* $\mathcal{L}$, which outputs $\mathbf{x}_t$, the 1-hot-vector encoding of w_t.

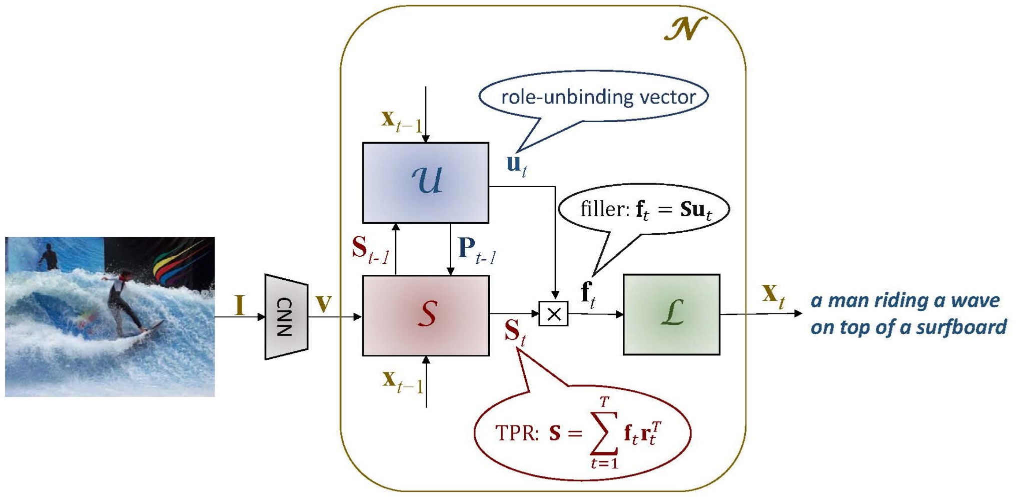


Figure 2: Architecture of TPGN, a TPR-capable generation network. "⊠" = matrix-vector product.

A crucial property of TPR is that unbinding is effected by the matrix-vector product. The key operation in generating w_t is thus the unbinding of r_t within $\mathbf{S}$, which amounts to simply:

$$\mathbf{S}_t \mathbf{u_t} = \mathbf{f_t}. \tag{5}$$

This matrix-vector product is denoted "⊠" in Fig. 2.

Thus the system $\mathcal{N}$ of Fig. 2 is TPR-capable. This is what we propose as the Tensor-Product Generation Network (TPGN) architecture. The learned representation $\mathbf{S}$ will not be proven to literally be a TPR, but by analyzing the unbinding vectors $\mathbf{u}_t$ the network learns, we will gain insight into the process by which the learned matrix $\mathbf{S}$ gives rise to the generated caption.

What type of roles might the unbinding vectors be unbinding? A TPR for a caption could in principle be built upon *positional roles*, *syntactic/semantic roles*, or some combination of the two. In the caption ***a man standing in a room with a suitcase***, the initial *a* and *man* might respectively occupy the positional roles of POS(ITION)_1 and POS_2; *standing* might occupy the syntactic role of VERB; *in* the role of SPATIAL-P(REPOSITION); while *a room with a suitcase* might fill a 5-role schema DET(ERMINER)_1 N(OUN)_1 P DET_2 N_2. For evidence that our network learns just this kind of hybrid role decomposition, see [12].

What form of information does the sentence-encoding subnetwork $\mathcal{S}$ need to encode in $\mathbf{S}$? Continuing with the example of the previous paragraph, $\mathbf{S}$ needs to be some approximation to the TPR summing several symbol/role binding matrices. In one of these bindings, a symbol vector $\mathbf{f}_a$ — which the lexical subnetwork $\mathcal{L}$ will map to the article *a* — is bound (via the outer product) to a role vector $\mathbf{r}_{\text{POS}_1}$ which is the dual of the first unbinding vector produced by the unbinding subnetwork $\mathcal{U}$: $\mathbf{u}_{\text{POS}_1}$. In the first iteration of generation the model computes $\mathbf{S}_1 \mathbf{u}_{\text{POS}_1} = \mathbf{f}_a$, which $\mathcal{L}$ then maps to *a*. Analogously, another binding approximately contained in $\mathbf{S}_2$ is $\mathbf{f}_{man}\mathbf{r}_{\text{POS}_2}^\top$. There are corresponding bindings for the remaining words of the caption; these employ syntactic/semantic roles. One example is $\mathbf{f}_{standing}\mathbf{r}_V^\top$. At iteration 3, $\mathcal{U}$ decides the next word should be a verb, so it generates the unbinding vector $\mathbf{u}_V$ which when multiplied by the current output of $\mathcal{S}$, the matrix $\mathbf{S}_3$, yields a filler vector $\mathbf{f}_{standing}$ which $\mathcal{L}$ maps to the output *standing*. $\mathcal{S}$ decided the caption should deploy *standing* as a verb and included in $\mathbf{S}$ the binding $\mathbf{f}_{standing}\mathbf{r}_V^\top$. It similarly decided the caption should deploy *in* as a spatial preposition, including in $\mathbf{S}$ the binding $\mathbf{f}_{in}\mathbf{r}_{\text{SPATIAL-P}}^\top$; and so on for the other words in their respective roles in the caption.

The unbinding subnetwork $\mathcal{U}$ and the sentence-encoding network $\mathcal{S}$ of Fig. 2 are each implemented as (1-layer, 1-directional) LSTMs; the lexical subnetwork $\mathcal{L}$ is implemented as a linear transformation followed by a softmax operation.

3.2 Experimental results

3.2.1 Evaluation of the image captioning system

To evaluate the performance of our proposed architecture, we use the COCO dataset [5]. For the CNN of Fig. 2, we used ResNet-152 [9], pretrained on the ImageNet dataset. The feature vector $\mathbf{v}$ has 2048 dimensions. Word embedding vectors in $\mathbf{W}_e$ are downloaded from the web [19]. The model is implemented in TensorFlow [1] with the default settings for random initialization and optimization by backpropagation. In our experiments, the dimension of $\mathbf{S}_t$ is 625×625; the vocabulary size $V = 8,791$; the dimension of $\mathbf{u}_t$ and $\mathbf{f}_t$ is 625.

Table 3: Performance of the proposed TPGN model on the COCO dataset.

Methods	METEOR	BLEU-1	BLEU-2	BLEU-3	BLEU-4	CIDEr
NIC [29]	–	0.666	0.451	0.304	0.203	–
CNN-LSTM	0.238	0.698	0.525	0.390	0.292	0.889
TPGN	**0.243**	**0.709**	**0.539**	**0.406**	**0.305**	**0.909**

The main evaluation results on the MS COCO dataset are given in Table 3, which reports the widely-used BLEU [17], METEOR [2], and CIDEr [28] metrics. In evaluation, our baseline is the widely used CNN-LSTM captioning method originally proposed in [29] (first line of Table 3). We also re-implemented the model using the latest ResNet features and report the results in the second line of Table 3. Our re-implementation of the CNN-LSTM method matches the performance reported in [6], showing that the baseline is a state-of-the-art implementation. As shown in Table 3, compared to the CNN-LSTM baseline, the proposed TPGN significantly outperforms the benchmark schemes in all metrics across the board. The improvement in BLEU-n is greater for greater n; TPGN particularly improves generation of longer subsequences. The results attest to the effectiveness of the TPGN architecture.

3.2.2 Interpretation of the learned roles

We now consider the extent to which the unbinding vectors that the TPGN system has learned to produce contain syntactic information. We assess this by determining how well classifiers can identify POS tags for words given the unbinding vectors used to generate those words; the gold standard for these tags are taken to be the output of the Stanford tagger [13]. This test is done on a variant of the architecture in which the visual-image input is replaced by a gold-standard caption, fed one word at a time into an LSTM to generate a vector which, like the feature vector input from the CNN, is used to initialize $\mathcal{S}$ and hence (ideally) re-generate the caption. We run this system with 5,000 sentences from the COCO test set as input, and obtain an unbinding vector $\mathbf{u}_t$ of each word $\mathbf{x}_t$ in the sentence produced by the TPGN system.

We designed a classifier for predicting the POS of each word $\mathbf{x}_t$. The classifier is a kernel support vector machine with stochastic gradient descent, using a radial basis function kernel. The input of the classifier is N_w unbinding vectors corresponding to a window of N_w words centered on the word to be classified. We used the unbinding vectors and the Stanford tagger's POS tags of 4,000 sentences for training, and the unbinding vectors of 1,000 sentences for testing. Thus in a sense we measure how close our system is to having the implicit syntactic knowledge that is possessed by the Stanford tagger.

Table 4: Predicting POS from learned unbinding vectors

Window size N_w	1	3	5	7	9	11	13
Precision	0.757	0.944	0.937	**0.946**	0.934	0.929	0.919
Recall	0.763	0.929	0.936	**0.942**	0.928	0.927	0.922
F-measure	0.760	0.937	0.936	**0.944**	0.931	0.928	0.921

Table 4 shows the results for classifying the POS of the words in the captions. It can be seen that, using the unbinding vector of a single word, we can classify the POS of the word with an accuracy of 76.3% (using the Stanford tagger as a reference). This means that a single unbinding vector contains important, but partial, grammatical information about the corresponding word. If the unbinding vectors of neighboring words are used, the accuracy of POS classification can be significantly increased to over 92%. The highest accuracy is achieved when the window size is 7; the F-score is 94.4%. This shows that the representations that TPGN learns — without any supervison by grammatical information — contain much of the rich syntactic information found in a supervised model such as the Stanford tagger.

For much further discussion of TPGN, see [12].

4 Related work

Architecture. In recent years, a number of DNNs have achieved notable success by reintroducing elements of symbolic computation as peripheral modules. This includes, e.g.: (i) the memory bank, a discrete set of addressed storage registers each holding a neural activation vector [10, 27, 30]; and (ii) the sequential program, a discrete sequence of steps, each selected from a discrete set of simple, approximately-discrete primitive operations [7, 14]. The discreteness in these peripheral modules is softened by continuous parameters with which they interface with the central controlling DNN; these 'attention' parameters modulate (i) the writing and reading operations with which information enters and exits a memory bank [4, 31]; and (ii) the extent to which inputs are passed to and outputs retrieved from the set of operations constituting a program [8]. The continuity of these parameters is of course crucial to enabling the overall system to be learnable by gradient-based optimization.

TPRNs explicitly, and TPGNs implicitly, constitute a different approach to reintroducing approximately symbolic representations and rule-based processing into neural network computation over continuous distributed representations. In computation with TPRs, the symbols and rules are internal to the DNN; there is no separation between a central network controller and peripheral quasi-discrete modules. Items in memories are distributed representations that are combined by addition/superposition rather than by being slotted into external discrete locations. Computation over TPRs is massively parallel [24].

Interpretation. Most methods of interpreting the internal representations of DNNs do so through the input and output representations of DNNs which are by necessity interpretable: these are where the DNN must interface with our description of its problem domain. An internal neuron may be interpreted by looking at the (interpretable) input patterns that activate it, or the (interpretable) output patterns that it activates (e.g., [32]).

The method pursued in this paper, by contrast, interprets internal DNN states not via $I \to O$ behavior but via an abstract theory of the system's problem domain. In the case of a language processing problem, such theories are provided by theoretical linguistics and traditional, symbolic computational linguistics. The elements we have interpreted are TPR roles, and TPR fillers, which are distributed activation vectors incorporated into network representations via the summation of their tensor products; we have designed an architecture in which individual neurons localize the presence of such roles and fillers ($\mathbf{a}_{\mathrm{R}}{}^{(t)}$ and $\mathbf{a}_{\mathrm{S}}{}^{(t)}$). Our interpretation rests on the interrelations between activations of the roles and fillers selected to encode words-in-context with the lexical-semantic and grammatical properties attributed to those words-in-context by linguistic theories.

5 Conclusion

We introduced two architectures inspired by Tensor Product Representations (TPRs) — TPRN and TPGN — and evaluated these models on the important NLP tasks of machine reading comprehension and image-to-language generation. Our results show that compared to the widely adopted LSTM-based architecture, the proposed models demonstrate significant grammatical interpretability, with on-par or better performance on these challenging tasks.

References

[1] Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from `http://tensorflow.org/`.

[2] Banerjee, Satanjeev and Lavie, Alon. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pp. 65–72. Association for Computational Linguistics, 2005.

[3] Cho, Pyeong Whan and Smolensky, Paul. Bifurcation analysis of a Gradient Symbolic Computation model of incremental processing. In Papafragou, A., Grodner, D., Mirman, D., and Trueswell, J. C. (eds.), *Proceedings of the 38th Annual Conference of the Cognitive Science Society*, Austin, TX, 2016. Cognitive Science Society.

[4] Chorowski, Jan K, Bahdanau, Dzmitry, Serdyuk, Dmitriy, Cho, Kyunghyun, and Bengio, Yoshua. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pp. 577–585, 2015.

[5] COCO. Coco dataset for image captioning. `http://mscoco.org/dataset/#download`, 2017.

[6] Gan, Zhe, Gan, Chuang, He, Xiaodong, Pu, Yunchen, Tran, Kenneth, Gao, Jianfeng, Carin, Lawrence, and Deng, Li. Semantic compositional networks for visual captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[7] Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[8] Graves, Alex, Wayne, Greg, Reynolds, Malcolm, Harley, Tim, Danihelka, Ivo, Grabska-Barwińska, Agnieszka, Colmenarejo, Sergio Gómez, Grefenstette, Edward, Ramalho, Tiago, Agapiou, John, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[9] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[10] Henaff, Mikael, Weston, Jason, Szlam, Arthur, Bordes, Antoine, and LeCun, Yann. Tracking the world state with recurrent entity networks. In *6th International Conference for Learning Representations*, Toulon, France, 2017.

[11] Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[12] Huang, Qiuyuan, Smolensky, Paul, He, Xiaodong, and Deng, Li. Tensor Product Generation Networks. *arXiv preprint arXiv:1709.09118*, 2017.

[13] Manning, Christopher. Stanford parser. `https://nlp.stanford.edu/software/lex-parser.shtml`, 2017.

[14] Neelakantan, Arvind, Le, Quoc V, and Sutskever, Ilya. Neural programmer: Inducing latent programs with gradient descent. In *5th International Conference for Learning Representations*, San Juan, Puerto Rico, 2016.

[15] Palangi, Hamid, Deng, Li, Shen, Yelong, Gao, Jianfeng, He, Xiaodong, Chen, Jianshu, Song, Xinying, and Ward, Rabab. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(4):694–707, 2016.

[16] Palangi, Hamid, Smolensky, Paul, He, Xiaodong, and Deng, Li. Question-answering with grammatically-interpretable representations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, New Orleans, LA, 2018.

[17] Papineni, Kishore, Roukos, Salim, Ward, Todd, and Zhu, Wei-Jing. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318. Association for Computational Linguistics, 2002.

[18] Pennington, Jeffrey, Socher, Richard, and Manning, Christopher. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014.

[19] Pennington, Jeffrey, Socher, Richard, and Manning, Christopher. Stanford glove: Global vectors for word representation. `https://nlp.stanford.edu/projects/glove/`, 2017.

[20] Rajpurkar, Pranav, Zhang, Jian, Lopyrev, Konstantin, and Liang, Percy. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP*, Austin, Texas, USA, 2016. Available at `http://arxiv.org/abs/1606.05250`.

[21] Seo, Min Joon, Kembhavi, Aniruddha, Farhadi, Ali, and Hajishirzi, Hannaneh. Bidirectional attention flow for machine comprehension. In *5th International Conference for Learning Representations*, San Juan, Puerto Rico, 2016. Available at `https://arxiv.org/abs/1611.01603`.

[22] Smolensky, Paul. Tensor product variable binding and the representation of symbolic structures in connectionist networks. *Artificial Intelligence*, 46:159–216, 1990.

[23] Smolensky, Paul. Symbolic functions from neural computation. *Philosophical Transactions of the Royal Society — A: Mathematical, Physical and Engineering Sciences*, 370:3543–3569, 2012.

[24] Smolensky, Paul and Legendre, Geraldine. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*. The MIT Press, Cambridge, MA, 2006.

[25] Smolensky, Paul, Goldrick, Matthew, and Mathis, Donald. Optimization and quantization in gradient symbol systems: A framework for integrating the continuous and the discrete in cognition. *Cognitive Science*, 38:1102–1138, 2014.

[26] Socher, Richard, Huang, Eric H, Pennin, Jeffrey, Manning, Christopher D, and Ng, Andrew Y. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pp. 801–809, 2011.

[27] Sukhbaatar, Sainbayar, Weston, Jason, Fergus, Rob, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pp. 2440–2448, 2015.

[28] Vedantam, Ramakrishna, Lawrence Zitnick, C, and Parikh, Devi. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4566–4575, 2015.

[29] Vinyals, Oriol, Toshev, Alexander, Bengio, Samy, and Erhan, Dumitru. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3156–3164, 2015.

[30] Weston, Jason, Chopra, Sumit, and Bordes, Antoine. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[31] Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron, Salakhudinov, Ruslan, Zemel, Rich, and Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pp. 2048–2057, 2015.

[32] Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.