Tensor Product Generation Networks for Deep NLP Modeling

Qiuyuan Huang[1], **Paul Smolensky**[1], **Xiaodong He**[2], **Li Deng**[1]*, **Dapeng Wu**[3]
[1]Microsoft Research, WA, USA; [2]JD AI Research; [3]University of Florida, FL, USA
{qihua,psmo}@microsoft.com; xiaodong.he@jd.com; l.deng@ieee.org
dpwu@ufl.edu

Abstract

We present a new approach to the design of deep networks for natural language processing (NLP), based on the general technique of Tensor Product Representations (TPRs) for encoding and processing symbol structures in distributed neural networks. A network architecture — the ***Tensor Product Generation Network*** (***TPGN***) — is proposed which is capable in principle of carrying out TPR computation, but which uses unconstrained deep learning to design its internal representations. Instantiated in a model for image-caption generation, TPGN outperforms LSTM baselines when evaluated on the COCO dataset. The TPR-capable structure enables interpretation of internal representations and operations, which prove to contain considerable grammatical content. Our caption-generation model can be interpreted as generating sequences of grammatical categories and retrieving words by their categories from a plan encoded as a distributed representation.

1 Introduction

In this paper we introduce a new architecture for natural language processing (NLP). On what type of principles can a computational architecture be founded? It would seem a sound principle to require that the hypothesis space for learning which an architecture provides include network hypotheses that are independently known to be suitable for performing the target task. Our proposed architecture makes available to deep learning network configurations that perform natural language generation by use of *Tensor Product Representations* (TPRs) (Smolensky and Legendre, 2006). Whether learning will create TPRs is unknown in advance, but what we can say with certainty is that the hypothesis space being searched during learning includes TPRs as one appropriate solution to the problem.

TPRs are a general method for generating vector-space embeddings of complex symbol structures. Prior work has proved that TPRs enable powerful symbol processing to be carried out using neural network computation (Smolensky, 2012). This includes generating parse trees that conform to a grammar (Cho et al., 2017), although incorporating such capabilities into deep learning networks such as those developed here remains for future work. The architecture presented here relies on simpler use of TPRs to generate sentences; grammars are not explicitly encoded here.

We test the proposed architecture by applying it to image-caption generation (on the MS-COCO dataset, (COCO, 2017)). The results improve upon a baseline deploying a state-of-the-art LSTM architecture (Vinyals et al., 2015), and the TPR foundations of the architecture provide greater interpretability.

Section 2 of the paper reviews TPR. Section 3 presents the proposed architecture, the *Tensor Product Generation Network* (TPGN). Section 4 describes the particular model we study for image captioning, and Section 5 presents the experimental results. Importantly, what the model has learned is interpreted in Section 5.3. Section 6 discusses the relation of the new model to previous work and Section 7 concludes.

2 Review of tensor product representation

The central idea of TPRs (Smolensky, 1990) can be appreciated by contrasting the TPR for a word string with a bag-of-words (BoW) vector-space embedding. In a BoW embedding, the vector that encodes `Jay saw Kay` is the same as the one that encodes `Kay saw Jay`: $\mathbf{J} + \mathbf{K} + \mathbf{s}$ where

*LD is currently at Citadel.

$\mathbf{J}, \mathbf{K}, \mathbf{s}$ are respectively the vector embeddings of the words `Jay`, `Kay`, `saw`.

A TPR embedding that avoids this confusion starts by analyzing `Jay saw Kay` as the set $\{$`Jay`/SUBJ, `Kay`/OBJ, `saw`/VERB$\}$. (Other analyses are possible: see Section 3.) Next we choose an embedding in a vector space V_F for `Jay`, `Kay`, `saw` as in the BoW case: $\mathbf{J}, \mathbf{K}, \mathbf{s}$. Then comes the step unique to TPRs: we choose an embedding in a vector space V_R for the ***roles*** SUBJ, OBJ, VERB: $\mathbf{r}_{\text{SUBJ}}, \mathbf{r}_{\text{OBJ}}, \mathbf{r}_{\text{VERB}}$. Crucially, $\mathbf{r}_{\text{SUBJ}} \neq \mathbf{r}_{\text{OBJ}}$. Finally, the TPR for `Jay saw Kay` is the following vector in $V_F \otimes V_R$:

$$\mathbf{v}_{\texttt{Jay saw Kay}} = \mathbf{J} \otimes \mathbf{r}_{\text{SUBJ}} + \mathbf{K} \otimes \mathbf{r}_{\text{OBJ}} + \mathbf{s} \otimes \mathbf{r}_{\text{VERB}} \tag{1}$$

Each word is tagged with the role it fills in the sentence; `Jay` and `Kay` fill different roles.

This TPR avoids the BoW confusion: $\mathbf{v}_{\texttt{Jay saw Kay}} \neq \mathbf{v}_{\texttt{Kay saw Jay}}$ because $\mathbf{J} \otimes \mathbf{r}_{\text{SUBJ}} + \mathbf{K} \otimes \mathbf{r}_{\text{OBJ}} \neq \mathbf{J} \otimes \mathbf{r}_{\text{OBJ}} + \mathbf{K} \otimes \mathbf{r}_{\text{SUBJ}}$. In the terminology of TPRs, in `Jay saw Kay`, `Jay` is the *filler* of the role SUBJ, and $\mathbf{J} \otimes \mathbf{r}_{\text{SUBJ}}$ is the vector embedding of the *filler/role binding* `Jay`/SUBJ. In the vector space embedding, the binding operation is the tensor — or generalized outer — product $\otimes$; i.e., $\mathbf{J} \otimes \mathbf{r}_{\text{SUBJ}}$ is a tensor with 2 indices defined by: $[\mathbf{J} \otimes \mathbf{r}_{\text{SUBJ}}]_{\varphi\rho} \equiv [\mathbf{J}]_{\varphi}[\mathbf{r}_{\text{SUBJ}}]_{\rho}$.

The tensor product can be used recursively, which is essential for the TPR embedding of recursive structures such as trees and for the computation of recursive functions over TPRs. However, in the present context, recursion will not be required, in which case the tensor product can be regarded as simply the matrix outer product (which cannot be used recursively); we can regard $\mathbf{J} \otimes \mathbf{r}_{\text{SUBJ}}$ as the matrix product $\mathbf{J}\mathbf{r}_{\text{SUBJ}}^{\top}$. Then Equation 1 becomes

$$\mathbf{v}_{\texttt{Jay saw Kay}} = \mathbf{J}\mathbf{r}_{\text{SUBJ}}^{\top} + \mathbf{K}\mathbf{r}_{\text{OBJ}}^{\top} + \mathbf{s}\mathbf{r}_{\text{VERB}}^{\top} \tag{2}$$

Note that the set of matrices (or the set of tensors with any fixed number of indices) is a vector space; thus `Jay saw Kay` $\mapsto \mathbf{v}_{\texttt{Jay saw Kay}}$ is a vector-space embedding of the symbol structures constituting sentences. Whether we regard $\mathbf{v}_{\texttt{Jay saw Kay}}$ as a 2-index tensor or as a matrix, we can call it simply a 'vector' since it is an element of a vector space: in the context of TPRs, 'vector' is used in a general sense and should not be taken to imply a single-indexed array.

Crucial to the computational power of TPRs and to the architecture we propose here is the notion of *unbinding*. Just as an *outer* product — the tensor product — can be used to *bind* the vector embedding a filler `Jay` to the vector embedding a role SUBJ, $\mathbf{J} \otimes \mathbf{r}_{\text{SUBJ}}$ or $\mathbf{J}\mathbf{r}_{\text{SUBJ}}^{\top}$, so an *inner* product can be used to take the vector embedding a structure and *unbind* a role contained within that structure, yielding the symbol that fills the role.

In the simplest case of orthonormal role vectors $\mathbf{r}_i$, to unbind role SUBJ in `Jay saw Kay` we can compute the matrix-vector product: $\mathbf{v}_{\texttt{Jay saw Kay}}\mathbf{r}_{\text{SUBJ}} = \mathbf{J}$ (because $\mathbf{r}_i^{\top}\mathbf{r}_j = \delta_{ij}$ when the role vectors are orthonormal). A similar situation obtains when the role vectors are not orthonormal, provided they are not linearly dependent: for each role such as SUBJ there is an *unbinding vector* $\mathbf{u}_{\text{SUBJ}}$ such that $\mathbf{r}_i^{\top}\mathbf{u}_j = \delta_{ij}$ so we get: $\mathbf{v}_{\texttt{Jay saw Kay}}\mathbf{u}_{\text{SUBJ}} = \mathbf{J}$. A role vector such as $\mathbf{r}_{\text{SUBJ}}$ and its unbinding vector $\mathbf{u}_{\text{SUBJ}}$ are said to be *duals* of each other. (If R is the matrix in which each column is a role vector $\mathbf{r}_j$, then R is invertible when the role vectors are linearly independent; then the unbinding vectors $\mathbf{u}_i$ are the rows of R^{-1}. When the $\mathbf{r}_j$ are orthonormal, $\mathbf{u}_i = \mathbf{r}_i$. Replacing the matrix inverse with the pseudo-inverse allows approximate unbinding if the role vectors are linearly dependent.)

We can now see how TPRs can be used to generate a sentence one word at a time. We start with the TPR for the sentence, e.g., $\mathbf{v}_{\texttt{Jay saw Kay}}$. From this vector we unbind the role of the first word, which is SUBJ: the embedding of the first word is thus $\mathbf{v}_{\texttt{Jay saw Kay}}\mathbf{u}_{\text{SUBJ}} = \mathbf{J}$, the embedding of `Jay`. Next we take the TPR for the sentence and unbind the role of the second word, which is VERB: the embedding of the second word is then $\mathbf{v}_{\texttt{Jay saw Kay}}\mathbf{u}_{\text{VERB}} = \mathbf{s}$, the embedding of `saw`. And so on.

To accomplish this, we need two representations to generate the t^{th} word: (i) the TPR of the sentence, $\mathbf{S}$ (or of the string of not-yet-produced words, $\mathbf{S}_t$) and (ii) the unbinding vector for the t^{th} word, $\mathbf{u}_t$. The architecture we propose will therefore be a recurrent network containing two subnetworks: (i) a subnet $\mathcal{S}$ hosting the representation $\mathbf{S}_t$, and a (ii) a subnet $\mathcal{U}$ hosting the unbinding vector $\mathbf{u}_t$. This is shown in Fig. 1.

3 A TPR-capable generation architecture

As Fig. 1 shows, the proposed Tensor Product Generation Network architecture (the dashed box labeled $\mathcal{N}$) is designed to support the technique

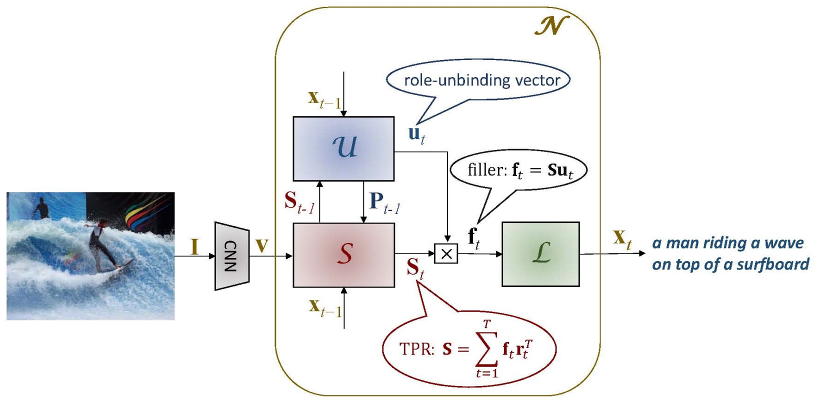

Figure 1: Architecture of TPGN, a TPR-capable generation network. "⊠" denotes the matrix-vector product.

for generation just described: the architecture is *TPR-capable*. There is a *sentence-encoding subnetwork* $\mathcal{S}$ which *could* host a TPR of the sentence to be generated, and an *unbinding subnetwork* $\mathcal{U}$ which *could* output a sequence of unbinding vectors $\mathbf{u}_t$; at time t, the embedding $\mathbf{f}_t$ of the word produced, $\mathtt{x}_t$, could then be extracted from $\mathbf{S}_t$ via the matrix-vector product (shown in the figure by "⊠"): $\mathbf{f}_t = \mathbf{S}_t\mathbf{u}_t$. The lexical-decoding subnetwork $\mathcal{L}$ converts the embedding vector $\mathbf{f}_t$ to the 1-hot vector $\mathbf{x}_t$ corresponding to the word $\mathtt{x}_t$.

Unlike some other work (Palangi et al., 2017), TPGN is not constrained to literally learn TPRs. The representations that will actually be housed in $\mathcal{S}$ and $\mathcal{U}$ are determined by end-to-end deep learning on a task: the bubbles in Fig. 1 show what *would* be the meanings of $\mathbf{S}_t, \mathbf{u}_t$ and $\mathbf{f_t}$ if an actual TPR scheme were instantiated in the architecture. The learned representations $\mathbf{S}_t$ will not be proven to literally be TPRs, but by analyzing the unbinding vectors $\mathbf{u}_t$ the network learns, we will gain insight into the process by which the learned matrices $\mathbf{S}_t$ give rise to the generated sentence.

The task studied here is image captioning; Fig. 1 shows that the input to this TPGN model is an image, preprocessed by a CNN which produces the initial representation in $\mathcal{S}$, $\mathbf{S}_0$. This vector $\mathbf{S}_0$ drives the entire caption-generation process: it 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.)

The two subnets $\mathcal{S}$ and $\mathcal{U}$ are mutually-connected LSTMs (Hochreiter and Schmidhuber, 1997): see Fig. 2. The internal hidden state of $\mathcal{U}$, $\mathbf{p}_t$, is sent as input to $\mathcal{S}$; $\mathcal{U}$ also produces output, the unbinding vector $\mathbf{u}_t$. The internal hidden state of $\mathcal{S}$, $\mathbf{S}_t$, is sent as input to $\mathcal{U}$, and also produced as output. As stated above, these two outputs are multiplied together to produce the embedding vector $\mathbf{f}_t = \mathbf{S}_t\mathbf{u}_t$ of the output word $\mathtt{x}_t$. Furthermore, the 1-hot encoding $\mathbf{x}_t$ of $\mathtt{x}_t$ is fed back at the next time step to serve as input to both $\mathcal{S}$ and $\mathcal{U}$.

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$_2$. In fact we will provide evidence in Sec. 5.3.2 that our network learns just this kind of hybrid role decomposition; further evidence for these particular roles is presented elsewhere.

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 filler/role binding matrices. In one of these bindings, a filler vector $\mathbf{f}_{\mathtt{a}}$ — 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 $\mathbf{S}_1\mathbf{u}_{\text{POS}_1} = \mathbf{f}_{\mathtt{a}}$, which $\mathcal{L}$ then maps to `a`. Analogously, another binding approximately contained in $\mathbf{S}_2$ is $\mathbf{f}_{\mathtt{man}}\mathbf{r}_{\text{POS}_2}^{\top}$. There are corresponding approximate bindings for the remaining words of the caption; these employ syntactic/semantic roles. One example is $\mathbf{f}_{\texttt{standing}}\mathbf{r}_V^\top$. 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 $\mathbf{f}_{\texttt{standing}}$ which $\mathcal{L}$ maps to the output `standing`. $\mathcal{S}$ decided the caption should deploy `standing` as a verb and included in $\mathbf{S}$ an approximation to the binding $\mathbf{f}_{\texttt{standing}}\mathbf{r}_V^\top$. It similarly decided the caption should deploy `in` as a spatial preposition, approximately including in $\mathbf{S}$ the binding $\mathbf{f}_{\texttt{in}}\mathbf{r}_{\text{SPATIAL-P}}^\top$; and so on for the other words in their respective roles in the caption.

4 System Description

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

In the equations below, the LSTM variables internal to the $\mathcal{S}$ subnet are indexed by 1 (e.g., the forget-, input-, and output-gates are respectively $\hat{\mathbf{f}}_1, \hat{\mathbf{i}}_1, \hat{\mathbf{o}}_1$) while those of the unbinding subnet $\mathcal{U}$ are indexed by 2.

Thus the state updating equations for $\mathcal{S}$ are, for $t = 1, \cdots, T$ = caption length:

$$\hat{\mathbf{f}}_{1,t} = \sigma_g(\mathbf{W}_{1,f}\mathbf{p}_{t-1} - \mathbf{D}_{1,f}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{1,f}\hat{\mathbf{S}}_{t-1}) \quad (3)$$

$$\hat{\mathbf{i}}_{1,t} = \sigma_g(\mathbf{W}_{1,i}\mathbf{p}_{t-1} - \mathbf{D}_{1,i}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{1,i}\hat{\mathbf{S}}_{t-1}) \quad (4)$$

$$\hat{\mathbf{o}}_{1,t} = \sigma_g(\mathbf{W}_{1,o}\mathbf{p}_{t-1} - \mathbf{D}_{1,o}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{1,o}\hat{\mathbf{S}}_{t-1}) \quad (5)$$

$$\mathbf{g}_{1,t} = \sigma_h(\mathbf{W}_{1,c}\mathbf{p}_{t-1} - \mathbf{D}_{1,c}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{1,c}\hat{\mathbf{S}}_{t-1}) \quad (6)$$

$$\mathbf{c}_{1,t} = \hat{\mathbf{f}}_{1,t} \odot \mathbf{c}_{1,t-1} + \hat{\mathbf{i}}_{1,t} \odot \mathbf{g}_{1,t} \quad (7)$$

$$\hat{\mathbf{S}}_t = \hat{\mathbf{o}}_{1,t} \odot \sigma_h(\mathbf{c}_{1,t}) \quad (8)$$

Here $\hat{\mathbf{f}}_{1,t}, \hat{\mathbf{i}}_{1,t}, \hat{\mathbf{o}}_{1,t}, \mathbf{g}_{1,t}, \mathbf{c}_{1,t}, \hat{\mathbf{S}}_t \in \mathbb{R}^{d\times d}$, $\mathbf{p}_t \in \mathbb{R}^d$; $\sigma_g(\cdot)$ is the (element-wise) logistic sigmoid function; $\sigma_h(\cdot)$ is the hyperbolic tangent function; the operator $\odot$ denotes the Hadamard (element-wise) product; $\mathbf{W}_{1,f}, \mathbf{W}_{1,i}, \mathbf{W}_{1,o}, \mathbf{W}_{1,c} \in \mathbb{R}^{(d\times d)\times d}$, $\mathbf{D}_{1,f}$, $\mathbf{D}_{1,i}$, $\mathbf{D}_{1,o}$, $\mathbf{D}_{1,c} \in \mathbb{R}^{(d\times d)\times d}$, $\mathbf{U}_{1,f}$, $\mathbf{U}_{1,i}$, $\mathbf{U}_{1,o}$, $\mathbf{U}_{1,c} \in \mathbb{R}^{(d\times d)\times(d\times d)}$. For clarity, biases — included throughout the model — are omitted from all equations in this paper. The initial state $\hat{\mathbf{S}}_0$ is initialized by:

$$\hat{\mathbf{S}}_0 = \mathbf{C}_s(\mathbf{v} - \bar{\mathbf{v}}) \quad (9)$$

where $\mathbf{v} \in \mathbb{R}^{2048}$ is the vector of visual features extracted from the current image by ResNet (Gan et al., 2017) and $\bar{\mathbf{v}}$ is the mean of all such vectors; $\mathbf{C}_s \in \mathbb{R}^{(d\times d)\times 2048}$. On the output side, $\mathbf{x}_t \in \mathbb{R}^V$ is a 1-hot vector with dimension equal to the size of the caption vocabulary, V, and $\mathbf{W}_e \in \mathbb{R}^{d\times V}$ is a word embedding matrix, the i-th column of which is the embedding vector of the i-th word in the vocabulary; it is obtained by the Stanford GLoVe algorithm with zero mean (Pennington et al., 2017). $\mathbf{x}_0$ is initialized as the one-hot vector corresponding to a "start-of-sentence" symbol.

For $\mathcal{U}$ in Fig. 1, the state updating equations are:

$$\hat{\mathbf{f}}_{2,t} = \sigma_g(\hat{\mathbf{S}}_{t-1}\mathbf{w}_{2,f} - \mathbf{D}_{2,f}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{2,f}\mathbf{p}_{t-1}) \quad (10)$$

$$\hat{\mathbf{i}}_{2,t} = \sigma_g(\hat{\mathbf{S}}_{t-1}\mathbf{w}_{2,i} - \mathbf{D}_{2,i}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{2,i}\mathbf{p}_{t-1}) \quad (11)$$

$$\hat{\mathbf{o}}_{2,t} = \sigma_g(\hat{\mathbf{S}}_{t-1}\mathbf{w}_{2,o} - \mathbf{D}_{2,o}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{2,o}\mathbf{p}_{t-1}) \quad (12)$$

$$\mathbf{g}_{2,t} = \sigma_h(\hat{\mathbf{S}}_{t-1}\mathbf{w}_{2,c} - \mathbf{D}_{2,c}\mathbf{W}_e\mathbf{x}_{t-1} + \mathbf{U}_{2,c}\mathbf{p}_{t-1}) \quad (13)$$

$$\mathbf{c}_{2,t} = \hat{\mathbf{f}}_{2,t} \odot \mathbf{c}_{2,t-1} + \hat{\mathbf{i}}_{2,t} \odot \mathbf{g}_{2,t} \quad (14)$$

$$\mathbf{p}_t = \hat{\mathbf{o}}_{2,t} \odot \sigma_h(\mathbf{c}_{2,t}) \quad (15)$$

Here $\mathbf{w}_{2,f}, \mathbf{w}_{2,i}, \mathbf{w}_{2,o}, \mathbf{w}_{2,c} \in \mathbb{R}^d$, $\mathbf{D}_{2,f}$, $\mathbf{D}_{2,i}$, $\mathbf{D}_{2,o}$, $\mathbf{D}_{2,c} \in \mathbb{R}^{d\times d}$, and $\mathbf{U}_{2,f}$, $\mathbf{U}_{2,i}$, $\mathbf{U}_{2,o}$, $\mathbf{U}_{2,c}$ $\in \mathbb{R}^{d\times d}$. The initial state $\mathbf{p}_0$ is the zero vector.

The dimensionality of the crucial vectors shown in Fig. 1, $\mathbf{u}_t$ and $\mathbf{f}_t$, is increased from $d\times 1$ to $d^2\times 1$ as follows. A block-diagonal $d^2 \times d^2$ matrix $\mathbf{S}_t$ is created by placing d copies of the $d \times d$ matrix $\hat{\mathbf{S}}_t$ as blocks along the principal diagonal. This matrix is the output of the sentence-encoding subnetwork $\mathcal{S}$. Now the 'filler vector' $\mathbf{f}_t \in \mathbb{R}^{d^2}$ — 'unbound' from the sentence representation $\mathbf{S}_t$ with the 'unbinding vector' $\mathbf{u}_t$ — is obtained by Eq. (16).

$$\mathbf{f}_t = \mathbf{S}_t\mathbf{u}_t \quad (16)$$

Here $\mathbf{u}_t \in \mathbb{R}^{d^2}$, the output of the unbinding subnetwork $\mathcal{U}$, is computed as in Eq. (17), where $\mathbf{W}_u \in \mathbb{R}^{d^2\times d}$ is $\mathcal{U}$'s output weight matrix.

$$\mathbf{u}_t = \sigma_h(\mathbf{W}_u\mathbf{p}_t) \quad (17)$$

Finally, the lexical subnetwork $\mathcal{L}$ produces a decoded word $\mathbf{x}_t \in \mathbb{R}^V$ by

$$\mathbf{x}_t = \sigma_s(\mathbf{W}_x\mathbf{f}_t) \quad (18)$$

where $\sigma_s(\cdot)$ is the softmax function and $\mathbf{W}_x \in \mathbb{R}^{V\times d^2}$ is the overall output weight matrix. Since $\mathbf{W}_x$ plays the role of a word de-embedding matrix, we can set

$$\mathbf{W}_x = (\mathbf{W}_e)^\top \quad (19)$$

where $\mathbf{W}_e$ is the word-embedding matrix. Since $\mathbf{W}_e$ is pre-defined, we directly set $\mathbf{W}_x$ by Eq. (19) without training $\mathcal{L}$ through Eq. (18). Note that $\mathcal{S}$ and $\mathcal{U}$ are learned jointly through end-to-end training as shown in Algorithm 1.

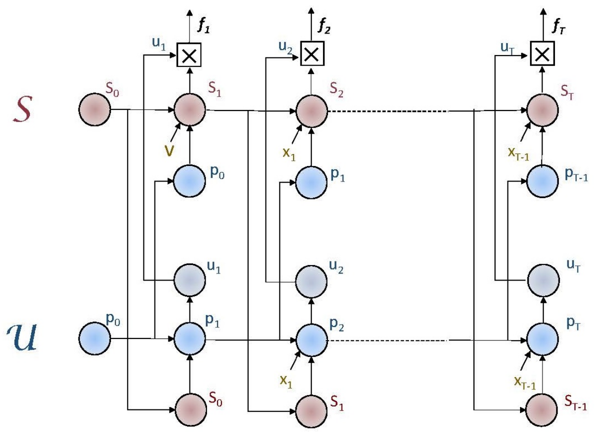

Figure 2: The sentence-encoding subnet $\mathcal{S}$ and the unbinding subnet $\mathcal{U}$ are inter-connected LSTMs; $\mathbf{v}$ encodes the visual input while the $\mathbf{x}_t$ encode the words of the output caption.

Algorithm 1 End-to-end training of $\mathcal{S}$ and $\mathcal{U}$

Input: Image feature vector $\mathbf{v}^{(i)}$ and corresponding caption $\mathbf{X}^{(i)} = [\mathbf{x}_1^{(i)}, \cdots, \mathbf{x}_T^{(i)}]$ $(i = 1, \cdots, N)$, where N is the total number of samples.
Output: $\mathbf{W}_{1,f}, \mathbf{W}_{1,i}, \mathbf{W}_{1,o}, \mathbf{W}_{1,c}, \mathbf{C}_s, \mathbf{D}_{1,f}, \mathbf{D}_{1,i}, \mathbf{D}_{1,o}, \mathbf{D}_{1,c}, \mathbf{U}_{1,f}, \mathbf{U}_{1,i}, \mathbf{U}_{1,o}, \mathbf{U}_{1,c}, \mathbf{w}_{2,f}, \mathbf{w}_{2,i}, \mathbf{w}_{2,o}, \mathbf{w}_{2,c}, \mathbf{D}_{2,f}, \mathbf{D}_{2,i}, \mathbf{D}_{2,o}, \mathbf{D}_{2,c}, \mathbf{U}_{2,f}, \mathbf{U}_{2,i}, \mathbf{U}_{2,o}, \mathbf{U}_{2,c}, \mathbf{W}_u, \mathbf{W}_x$.

```
1:  Initialize $\mathbf{S}_0$ by (9);
2:  Initialize $\mathbf{x}_0$ as the one-hot vector corresponding to the start-of-sentence symbol;
3:  Initialize $\mathbf{p}_0$ as the zero vector;
4:  Randomly initialize weights $\mathbf{W}_{1,f}, \mathbf{W}_{1,i}, \mathbf{W}_{1,o}, \mathbf{W}_{1,c}, \mathbf{C}_s, \mathbf{D}_{1,f}, \mathbf{D}_{1,i}, \mathbf{D}_{1,o}, \mathbf{D}_{1,c}, \mathbf{U}_{1,f}, \mathbf{U}_{1,i}, \mathbf{U}_{1,o}, \mathbf{U}_{1,c}, \mathbf{w}_{2,f}, \mathbf{w}_{2,i}, \mathbf{w}_{2,o}, \mathbf{w}_{2,c}, \mathbf{D}_{2,f}, \mathbf{D}_{2,i}, \mathbf{D}_{2,o}, \mathbf{D}_{2,c}, \mathbf{U}_{2,f}, \mathbf{U}_{2,i}, \mathbf{U}_{2,o}, \mathbf{U}_{2,c}, \mathbf{W}_u, \mathbf{W}_x$;
5:  for $n$ from 1 to $N$ do
6:      for $t$ from 1 to $T$ do
7:          Calculate (3) – (8) to obtain $\mathbf{S}_t$;
8:          Calculate (10) – (15) to obtain $\mathbf{p}_t$;
9:          Calculate (17) to obtain $\mathbf{u}_t$;
10:         Calculate (16) to obtain $\mathbf{f}_t$;
11:         Calculate (18) to obtain $\mathbf{x}_t$;
12:         Update weights $\mathbf{W}_{1,f}, \mathbf{W}_{1,i}, \mathbf{W}_{1,o}, \mathbf{W}_{1,c}, \mathbf{C}_s, \mathbf{D}_{1,f}, \mathbf{D}_{1,i}, \mathbf{D}_{1,o}, \mathbf{D}_{1,c}, \mathbf{U}_{1,f}, \mathbf{U}_{1,i}, \mathbf{U}_{1,o}, \mathbf{U}_{1,c}, \mathbf{w}_{2,f}, \mathbf{w}_{2,i}, \mathbf{w}_{2,o}, \mathbf{w}_{2,c}, \mathbf{D}_{2,f}, \mathbf{D}_{2,i}, \mathbf{D}_{2,o}, \mathbf{D}_{2,c}, \mathbf{U}_{2,f}, \mathbf{U}_{2,i}, \mathbf{U}_{2,o}, \mathbf{U}_{2,c}, \mathbf{W}_u$
            by the back-propagation algorithm;
13:     end for
14: end for
```

5 Experimental results

5.1 Dataset

To evaluate the performance of our proposed model, we use the COCO dataset (COCO, 2017). The COCO dataset contains 123,287 images, each of which is annotated with at least 5 captions. We use the same pre-defined splits as in (Karpathy and Fei-Fei, 2015; Gan et al., 2017): 113,287 images for training, 5,000 images for validation, and 5,000 images for testing. We use the same vocabulary as that employed in (Gan et al., 2017), which consists of 8,791 words.

5.2 Evaluation

For the CNN of Fig. 1, we used ResNet-152 (He et al., 2016), 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 (Pennington et al., 2017). The model is implemented in TensorFlow (Abadi et al., 2015) with the default settings for random initialization and optimization by backpropagation.

In our experiments, we choose $d = 25$ (where d is the dimension of vector $\mathbf{p}_t$). The dimension of $\mathbf{S}_t$ is 625×625 (while $\hat{\mathbf{S}}_t$ is 25×25); the vocabulary size $V = 8,791$; the dimension of $\mathbf{u}_t$ and $\mathbf{f}_t$ is $d^2 = 625$.

The main evaluation results on the MS COCO dataset are reported in Table 1. The widely-used BLEU (Papineni et al., 2002), METEOR (Banerjee and Lavie, 2005), and CIDEr (Vedantam et al., 2015) metrics are reported in our quantitative evaluation of the performance of the proposed model. In evaluation, our baseline is the widely used CNN-LSTM captioning method originally proposed in (Vinyals et al., 2015). For comparison, we include results in that paper in the first line of Table 1. We also re-implemented the model using the latest ResNet features and report the results in the second line of Table 1. Our re-implementation of the CNN-LSTM method matches the performance reported in (Gan et al., 2017), showing that the baseline is a state-of-the-art implementation. For TPGN, we use parameter settings in a similar range to those in (Gan et al., 2017). TPGN has comparable, although slightly more, param-

Table 1: Performance of the proposed TPGN model on the COCO dataset.

Methods	METEOR	BLEU-1	BLEU-2	BLEU-3	BLEU-4	CIDEr
NIC (Vinyals et al., 2015)	0.237	0.666	0.461	0.329	0.246	0.855
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**

eters than the CNN-LSTM. The training time of TPGN is roughly 50% more than the CNN-LSTM model. The weights in TPGN are updated at every mini-batch; in the experiments, we use a batch size of 64 images. As shown in Table 1, compared to the CNN-LSTM baseline, the proposed TPGN appreciably 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.

It is worth mentioning that this paper is aimed at developing a Tensor Product Representation (TPR) inspired network to replace the core layers in an LSTM; therefore, it is directly comparable to an LSTM baseline. So in the experiments, we focus on comparison to a strong CNN-LSTM baseline. We acknowledge that more recent papers (Xu et al., 2017; Rennie et al., 2017; Yao et al., 2017; Lu et al., 2017; Gan et al., 2017) reported better performance on the task of image captioning. Performance improvements in these more recent models are mainly due to using better image features such as those obtained by Region-based Convolutional Neural Networks (R-CNN), or using reinforcement learning (RL) to directly optimize metrics such as CIDEr, or using more complex attention mechanisms (Gan et al., 2017) to provide a better context vector for caption generation, or using an ensemble of multiple LSTMs, among others. However, the LSTM is still playing a core role in these works and we believe improvement over the core LSTM, in both performance and interpretability, is still very valuable; that is why we compare the proposed TPGN with a state-of-the-art native LSTM (the second line of Table 1).

5.3 Interpretation of learned unbinding vectors

To get a sense of how the sentence encodings $\mathbf{S}_t$ learned by TPGN approximate TPRs, we now investigate the meaning of the role-unbinding vector $\mathbf{u}_t$ the model uses to unbind from $\mathbf{S}_t$ — via Eq. (16) — the filler vector $\mathbf{f}_t$ that produces — via Eq. (18) — the one-hot vector $\mathbf{x}_t$ of the t^{th} generated caption word. The meaning of an unbinding vector is the meaning of the role it unbinds. Interpreting the unbinding vectors reveals the meaning of the roles in a TPR that $\mathbf{S}$ approximates.

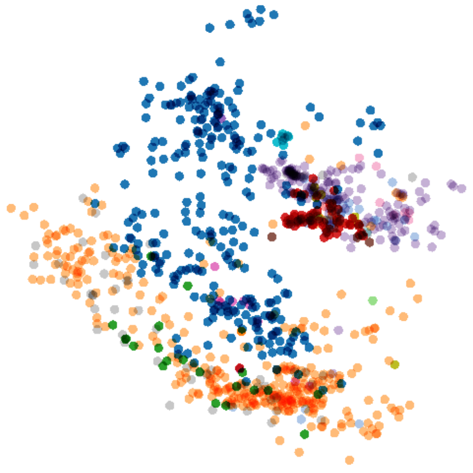

Figure 3: Unbinding vectors of 1000 words; different POS tags of words are represented by different colors.

5.3.1 Visualization of $\mathbf{u}_t$

We run the TPGN model with 5,000 test images as input, and obtain the unbinding vector $\mathbf{u}_t$ used to generate each word $\mathbf{x}_t$ in the caption of a test image. We plot 1,000 unbinding vectors $\mathbf{u}_t$, which correspond to the first 1,000 words in the resulting captions of these 5,000 test images. There are 17 parts of speech (POS) in these 1,000 words. The POS tags are obtained by the Stanford Parser (Manning, 2017).

We use the Embedding Projector in TensorBoard (Google, 2017) to plot 1,000 unbinding vectors $\mathbf{u}_t$ with a custom linear projection in TensorBoard to reduce 625 dimensions of $\mathbf{u}_t$ to 2 dimensions shown in Fig. 3 through Fig. 7.

Fig. 3 shows the unbinding vectors of 1000 words; different POS tags of words are represented by different colors. In fact, we can partition the 625-dim space of $\mathbf{u}_t$ into 17 regions, each of which contains 76.3% words of the same type of POS on average; i.e., each region is dominated by words of one POS type. This clearly indicates that *each unbinding vector contains important grammatical information about the word it generates*. As examples, Fig. 4 to Fig. 7 show the distribution of the unbinding vectors of nouns, verbs, adjectives, and prepositions, respectively. Furthermore, we show that the subject and the object of a sentence can be distinguished based on $\mathbf{u}_t$ in (Huang et al., 2018).

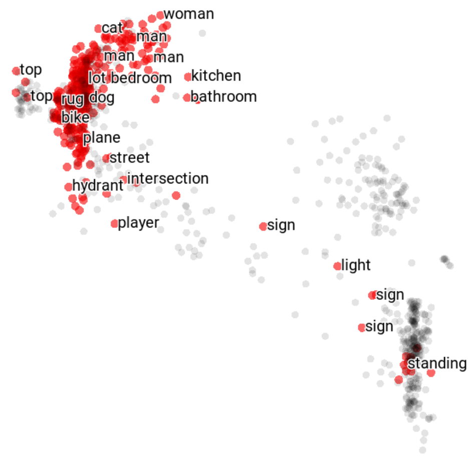

Figure 4: Unbinding vectors of 360 nouns in red and 640 words of other types of POS in grey.

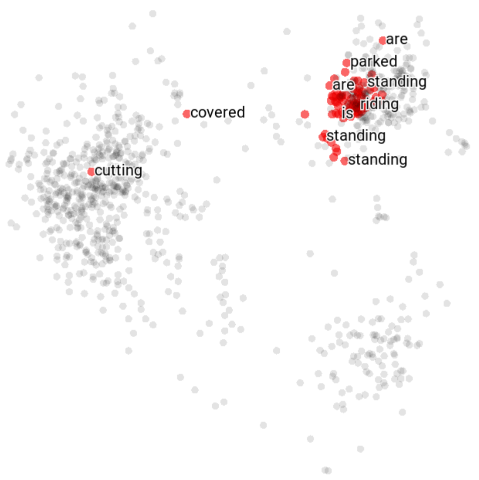

Figure 5: Unbinding vectors of 81 verbs in red and 919 words of other types of POS in grey.

5.3.2 Clustering of $\mathbf{u}_t$

Since the previous section indicates that there is a clustering structure for $\mathbf{u}_t$, in this section we partition $\mathbf{u}_t$ into N_u clusters and examine the grammar roles played by $\mathbf{u}_t$.

First, we run the trained TPGN model on the 113,287 training images, obtaining the role-

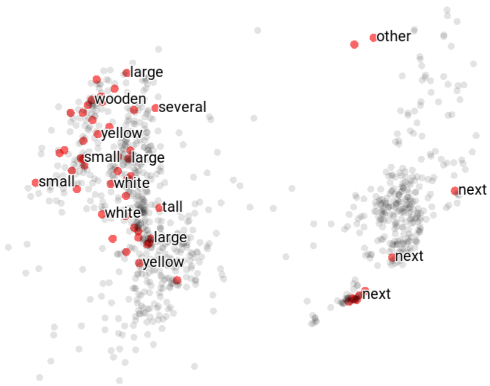

Figure 6: Unbinding vectors of 55 adjectives in red and 945 words of other types of POS in grey.

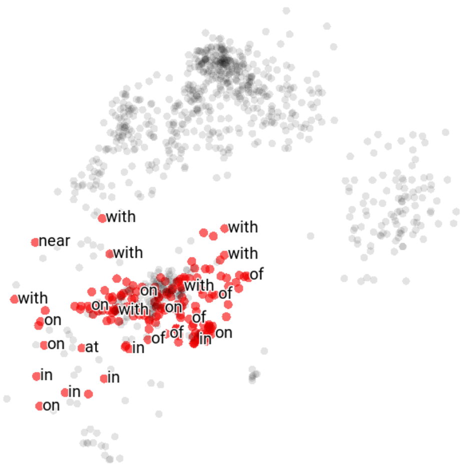

Figure 7: Unbinding vectors of 169 prepositions in red and 831 words of other types of POS in grey.

unbinding vector $\mathbf{u}_t$ used to generate each word $\mathbf{x}_t$ in the caption sentence. There are approximately 1.2 million $\mathbf{u}_t$ vectors over all the training images. We apply the K-means clustering algorithm to these vectors to obtain N_u clusters and the centroid $\boldsymbol{\mu}_i$ of each cluster i ($i = 0, \cdots, N_u - 1$).

Then, we run the TPGN model with 5,000 test images as input, and obtain the role vector $\mathbf{u}_t$ of each word $\mathbf{x}_t$ in the caption sentence of a test image. Using the nearest neighbor rule, we obtain the index i of the cluster that each $\mathbf{u}_t$ is assigned to.

The partitioning of the unbinding vectors $\mathbf{u}_t$ into $N_u = 2$ clusters exposes the most fundamental distinction made by the roles. We find that the vectors assigned to Cluster 1 generate words which are nouns, pronouns, indefinite and definite articles, and adjectives, while the vectors assigned to Cluster 0 generate verbs, prepositions, conjunctions, and adverbs. Thus Cluster 1 contains the noun-related words, Cluster 0 the verb-like words

Table 2: Conformity to N/V generalization ($N_u = 2$).

Category	N_w	N_r	P_c
Nouns	16683	16115	0.969
Pronouns	462	442	0.957
Indefinite articles	7248	7107	0.981
Definite articles	797	762	0.956
Adjectives	2543	2237	0.880
Verbs	3558	3409	0.958
Prepositions & conjunctions	8184	7859	0.960
Adverbs	13	8	0.615

Table 3: Interpretation of unbinding clusters ($N_u = 10$)

ID	Interpretation (proportion)
2	Position 1 (1.00)
3	Position 2 (1.00)
1	Noun (0.54), Determiner (0.43)
5	Determiner (0.50), Noun (0.19), Preposition (0.15)
7	Noun (0.88), Adjective (0.09)
9	Determiner (0.90), Noun (0.10)
0	Preposition (0.64), . (0.16), V (0.14)
4	Preposition: spatial (0.72) non-spatial (0.19)
6	Preposition (0.59), . (0.14)
8	Verb (0.37), Preposition (0.36), . (0.20)

(verbs, prepositions and conjunctions are all potentially followed by noun-phrase complements, for example). Cross-cutting this distinction is another dimension, however: the initial word in a caption (always a determiner) is sometimes generated with a Cluster 1 unbinding vector, sometimes with a Cluster 0 vector. Outside the caption-initial position, exceptions to the nominal/verbal $\sim$ Cluster 1/0 generalization are rare, as attested by the high rates of conformity to the generalization shown in Table 2.

Table 2 shows the likelihood of correctness of this 'N/V' generalization for the words in 5,000 sentences captioned for the 5,000 test images; N_w is the number of words in the category, N_r is the number of words conforming to the generalization, and $P_c = N_r/N_w$ is the proportion conforming. We use the Natural Language Toolkit (NLTK, 2017) to identify the part of speech of each word in the captions.

A similar analysis with $N_u = 10$ clusters reveals the results shown in Table 3; these results concern the first 100 captions, which were inspected manually to identify interpretable patterns. (More comprehensive results will be discussed elsewhere.)

The clusters can be interpreted as falling into 3 groups (see Table 3). Clusters 2 and 3 are clearly positional roles: every initial word is generated by a role-unbinding vector from Cluster 2, and such vectors are not used elsewhere in the string. The same holds for Cluster 3 and the second caption word.

For caption words after the second word, position is replaced by syntactic/semantic properties for interpretation purposes. The vector clusters aside from 2 and 3 generate words with a dominant grammatical category: for example, unbinding vectors assigned to the cluster 4 generate words that are 91% likely to be prepositions, and 72% likely to be spatial prepositions. Cluster 7 generates 88% nouns and 9% adjectives, with the remaining 3% scattered across other categories. As Table 3 shows, clusters 1, 5, 7, 9 are primarily nominal, and 0, 4, 6, and 8 primarily verbal. (Only cluster 5 spans the N/V divide.)

6 Related work

This work follows a great deal of recent caption-generation literature in exploiting end-to-end deep learning with a CNN image-analysis front end producing a distributed representation that is then used to drive a natural-language generation process, typically using RNNs (Mao et al., 2015; Vinyals et al., 2015; Devlin et al., 2015; Chen and Zitnick, 2015; Donahue et al., 2015; Karpathy and Fei-Fei, 2015; Kiros et al., 2014a,b; Xu et al., 2017; Rennie et al., 2017; Yao et al., 2017; Lu et al., 2017). Our grammatical interpretation of the structural roles of words in sentences makes contact with other work that incorporates deep learning into grammatically-structured networks (Tai et al., 2015; Kumar et al., 2016; Kong et al., 2017; Andreas et al., 2015; Yogatama et al., 2016; Maillard et al., 2017; Socher et al., 2010; Pollack, 1990). Here, the network is not itself structured to match the grammatical structure of sentences being processed; the structure is fixed, but is designed to support the learning of distributed representations that incorporate structure internal to the representations themselves — filler/role structure.

TPRs are also used in NLP in (Palangi et al., 2017) but there the representation of each individual input word is constrained to be a literal TPR filler/role binding. (The idea of using the outer product to construct internal representations was also explored in (Fukui et al., 2016).) Here, by contrast, the learned representations are not themselves constrained, but the global structure of the network is designed to display the somewhat abstract property of being TPR-capable: the architecture uses the TPR unbinding operation of the matrix-vector product to extract individual words for sequential output.

7 Conclusion

Tensor Product Representation (TPR) (Smolensky, 1990) is a general technique for constructing vector embeddings of complex symbol structures in such a way that powerful symbolic functions can be computed using hand-designed neural network computation. Integrating TPR with deep learning is a largely open problem for which the work presented here proposes a general approach: design deep architectures that are TPR-capable — TPR computation is within the scope of the capabilities of the architecture in principle. For natural language generation, we proposed such an architecture, the Tensor Product Generation Network (TPGN): it embodies the TPR operation of unbinding which is used to extract particular symbols (e.g., words) from complex structures (e.g., sentences). The architecture can be interpreted as containing a part that encodes a sentence and a part that selects one structural role at a time to extract from the sentence. We applied the approach to image-caption generation, developing a TPGN model that was evaluated on the COCO dataset, on which it outperformed LSTM baselines on a range of standard metrics. Unlike standard LSTMs, however, the TPGN model admits a level of interpretability: we can see which roles are being unbound by the unbinding vectors generated internally within the model. We find such roles contain considerable grammatical information, enabling POS tag prediction for the words they generate and displaying clustering by POS.

References

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

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2015. Deep compositional question answering with neural module networks. arxiv preprint. *arXiv preprint arXiv:1511.02799* 2.

Satanjeev Banerjee and Alon Lavie. 2005. 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*. Association for Computational Linguistics, pages 65–72.

Xinlei Chen and Lawrence Zitnick. 2015. Mind's eye: A recurrent visual representation for image caption generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 2422–2431.

Pyeong Whan Cho, Matthew Goldrick, and Paul Smolensky. 2017. Incremental parsing in a continuous dynamical system: Sentence processing in Gradient Symbolic Computation. *Linguistics Vanguard* 3. `DOI:10.1515/lingvan-2016-0105`.

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

Jacob Devlin, Hao Cheng, Hao Fang, Saurabh Gupta, Li Deng, Xiaodong He, Geoffrey Zweig, and Margaret Mitchell. 2015. Language models for image captioning: The quirks and what works. *arXiv preprint arXiv:1505.01809* .

Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 2625–2634.

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. 2016. Multimodal compact bilinear pooling for visual question answering and visual grounding. *arXiv preprint arXiv:1606.01847* .

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

Google. 2017. Embedding projector in tensorboard. `https://www.tensorflow.org/programmers_guide/embedding`.

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

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

Qiuyuan Huang, Li Deng, Dapeng Wu, Chang Liu, and Xiaodong He. 2018. Attentive tensor product learning for language generation and grammar parsing. *arXiv preprint arXiv:1802.07089* .

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 3128–3137.

Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. 2014a. Multimodal neural language models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. pages 595–603.

Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. 2014b. Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539* .

Lingpeng Kong, Chris Alberti, Daniel Andor, Ivan Bogatyy, and David Weiss. 2017. Dragnn: A transition-based framework for dynamically connected neural networks. *arXiv preprint arXiv:1703.04474* .

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*. pages 1378–1387.

Jiasen Lu, Caiming Xiong, Devi Parikh, and Richard Socher. 2017. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. volume 6.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2017. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *arXiv preprint arXiv:1705.09189* .

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

Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. 2015. Deep captioning with multimodal recurrent neural networks (m-rnn). In *Proceedings of International Conference on Learning Representations*.

NLTK. 2017. Natural language toolkit (nltk). `http://www.nltk.org`.

Hamid Palangi, Paul Smolensky, Xiaodong He, and Li Deng. 2017. Deep learning of grammatically-interpretable representations through question-answering. *arXiv preprint arXiv:1705.08432* .

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

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

Jordan B Pollack. 1990. Recursive distributed representations. *Artificial Intelligence* 46(1):77–105.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Paul Smolensky. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence* 46(1-2):159–216.

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

Paul Smolensky and Géraldine Legendre. 2006. *The harmonic mind: From neural computation to optimality-theoretic grammar. Volume 1: Cognitive architecture*. MIT Press.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*. pages 1–9.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075* .

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

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

Kaisheng Xu, Hanli Wang, and Pengjie Tang. 2017. Image captioning with deep lstm based on sequential residual. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*. pages 361–366.

Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. 2017. Boosting image captioning with attributes. In *Proceedings of International Conference on Computer Vision*.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2016. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1611.09100* .