RNNs IMPLICITLY IMPLEMENT TENSOR-PRODUCT REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent neural networks (RNNs) can learn continuous vector representations of symbolic structures such as sequences and sentences; these representations often exhibit linear regularities (analogies). Such regularities motivate our hypothesis that RNNs implicitly compile symbolic structures into tensor product representations (TPRs; Smolensky, 1990), which additively combine tensor products of vectors representing roles (e.g., sequence position) and vectors representing fillers (e.g., a particular word). To test this hypothesis, we introduce Tensor Product Decomposition Networks (TPDNs), which use TPRs to approximate existing vector representations. We demonstrate using synthetic data that TPDNs can successfully approximate linear and tree-based RNN autoencoder representations, suggesting that these representations exhibit highly interpretable compositional structure; we explore the settings that lead RNNs to induce such structure-sensitive representations. By contrast with these results, TPDN experiments with four standard sentence encoding models show that those sentence encodings can be largely approximated using bag-of-words representations, with only marginal improvements from more sophisticated structures. We conclude that TPDNs provide a powerful method for interpreting vector representations, and that standard RNNs can induce compositional sequence representations that are remarkably well approximated by TPRs; at the same time, existing training tasks for sentence representation learning may not be sufficient for inducing robust structural representations.

1 INTRODUCTION

Neural networks construct continuous vector representations of discrete elements such as words or sentences. In some cases, the spatial relationships between such vector representations have been argued to display geometric regularities that parallel plausible symbolic structures of the elements being represented (Mikolov et al. 2013; see Figure 1). These regularities may explain why these continuous representations succeed at tasks traditionally believed to require symbolic manipulation.

Analogical relationships such as those of Mikolov et al. (2013) are special cases of linearity properties shared by several general methods that were developed in the 1990s for designing compositional vectorial embeddings of symbolic structures. The most general of these is **tensor product representations** (TPRs; Smolensky 1990). Symbolic structures are first decomposed into **filler-role**

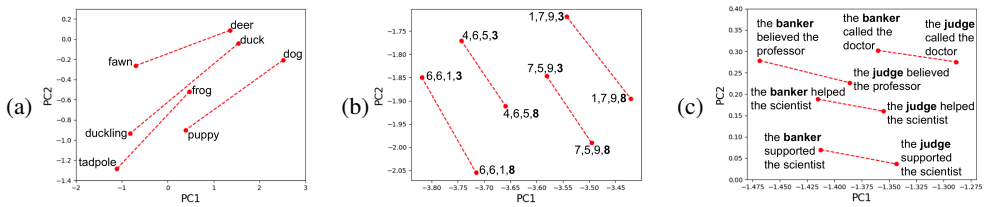


Figure 1: Plots of the first two principal components of (a) word embeddings (Pennington et al., 2014), (b) digit-sequence embeddings learned by an autoencoder (Section 2), and (c) sentences (InferSent: Conneau et al. 2017). All demonstrate systematicity in the learned vector spaces.

bindings; for example, to represent the sequence $[5, 2, 4]$, the filler 5 may be bound to the role of *the first element of the sequence*, the filler 2 may be bound to the role of *the second element of the sequence*, and so on. Each filler f_i and — crucially — each **role** r_i has a vector embedding; these two vectors are combined using their tensor product $f_i \otimes r_i$, and these tensor products are summed to produce the representation of the sequence: $\sum f_i \otimes r_i$. This linear combination can predict the linear relations between sequence representations illustrated in Figure 1.

In this article, we test the hypothesis that structure-sensitive recurrent neural network vector representations of sequences can be approximated as a linear combination of filler-role bindings, as in TPRs. Smolensky (1990) showed how to perform tensor product filler-role binding in the forward direction given known role and vector embeddings; we propose the Tensor Product Decomposition Network (TPDN) which inverts this process, taking a set of continuous vector representations to be analyzed and learning filler and role embeddings that best predict those vectors, given a particular hypothesis for the relevant set of roles (e.g., sequence indexes or structural positions in a parse tree).

To derive structure-sensitive representations, in Section 2 we look at a task that is driven by structure, not content: autoassociation of sequences of meaningless symbols, denoted by digits. The primary focus here is on sequential structure, although we also devise a version of the task that incorporates tree structure. For the intermediate representations of these autoencoders, TPDNs find excellent approximations that are TPRs.

In Section 3, we turn to prominent sentence-embedding models in the contemporary literature. It is an open question how structure-sensitive these representations are; to the degree that they are structure-sensitive, our hypothesis is that they can be approximated by TPRs. Here, TPDNs find less accurate approximations, but they also show that a TPR equivalent to a bag-of-words already provides a reasonably good approximation; these results call into question just how structure-sensitive these sentence representations are.

We therefore return to synthetic data in Section 4, exploring the model-architecture and training-task settings that are more likely to lead RNNs to induce structure-sensitive representations. Throughout these studies, TPDNs provide a powerful method for interpreting continuous vector representations and show that standard RNNs can induce compositional representations of sequences that are remarkably well approximated by TPRs.

1.1 The Tensor Product Decomposition Network

The Tensor Product Decomposition Network (TPDN), depicted in Figure 2c, learns a TPR that best approximates an existing set of vector encodings. While this architecture can be applied to any structured space, including embeddings of images or words, this work focuses on the application of TPDNs to sequences. The model is provided with a hypothesized role scheme, the dimensionality of the filler embeddings, and the dimensionality of the role embeddings. The elements of each sequence are assumed to be the fillers in that sequence's representation; for example, if the hypothesized roles are positional indexes counting from the end of the sequence, then the hypothesized filler-role bindings for $[5, 2, 4]$ would be (4:*last*, 2:*second-to-last*, 5:*third-to-last*).

The model then learns embeddings for these fillers and roles that minimize the distance between the TPRs generated from these embeddings and the existing encodings of the sequences. Before the comparison is performed, the tensor product (which is a matrix) is flattened into a vector, and a linear transformation M is applied (see Appendix B for an ablation study showing that this transformation, which was not a part of the original TPR proposal, is necessary). The overall function computed by the architecture is thus $M(\textit{flatten}(\sum_i r_i \otimes f_i))$.

PyTorch code implementing the TPDN model will be made available on GitHub.

2 Approximating RNN autoencoder representations

To establish the effectiveness of the TPDN at uncovering the structural representations used by RNNs, we first apply the TPDN to sequence-to-sequence networks trained on an autoencoding objective: they are expected to encode a sequence of digits and then decode that encoding to reproduce the same sequence (Figure 2a). In addition to testing the TPDN, this experiment also addresses a scientific question: do different architectures induce different representations?

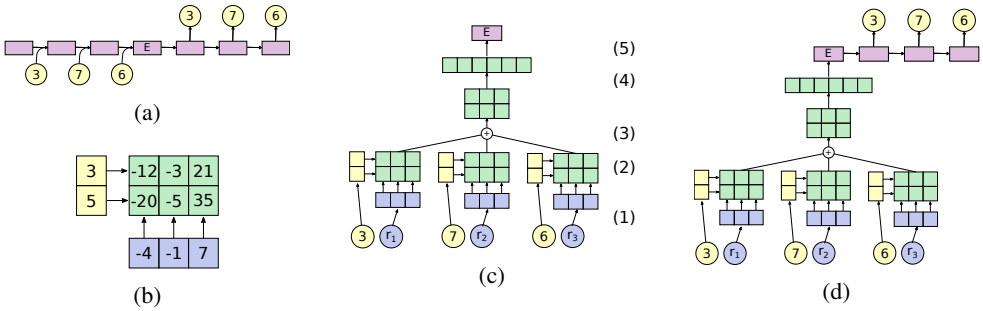


Figure 2: **(a)** A unidirectional sequence-to-sequence autoencoder. **(b)** The tensor product operation. **(c)** A TPDN trained to approximate the encoding E from the autoencoder: (1) The fillers and roles are embedded. (2) The fillers and roles are bound together using the tensor product. (3) The tensor products are summed. (4) The sum is flattened into a vector by concatenating the rows. (5) A linear transformation is applied to get the final encoding. **(d)** The architecture for evaluation: using the original autoencoder's decoder with the trained TPDN as the encoder.

2.1 Experimental setup

Digit sequences: The sequences consisted of the digits from 0 to 9. We randomly generated 50,000 unique sequences with lengths ranging from 1 to 6 inclusive and averaging 5.2; these sequences were divided into 40,000 training sequences, 5,000 development sequences, and 5,000 test sequences.

Architectures: For all sequence-to-sequence networks, we used gated recurrent units (GRUs, Cho et al. (2014)) as the recurrent units. We considered three types of network topologies: unidirectional, bidirectional, and tree-based, each of which can be used for the encoder or the decoder. The unidirectional encoders and decoders are the standard left-to-right sequence-to-sequence setup, used by Sutskever et al. (2014). The bidirectional encoder combines left-to-right and right-to-left unidirectional encoders (Schuster & Paliwal, 1997); for symmetry, we also create a bidirectional decoder, which has both a left-to-right and a right-to-left unidirectional decoder whose hidden states are concatenated to form bidirectional hidden states from which output predictions are made. Our final topology is tree-based RNNs (Pollack, 1990; Socher et al., 2010), specifically the Tree-GRU encoder of Chen et al. (2017) and the tree decoder of Chen et al. (2018). These architectures require a tree structure as part of their input; since digit sequences have no natural hierarchical structure, we generated a tree for each sequence using a deterministic algorithm that clusters digits based on their values (see Appendix C). To control for initialization effects, we trained five instances of each architecture with different random initializations.

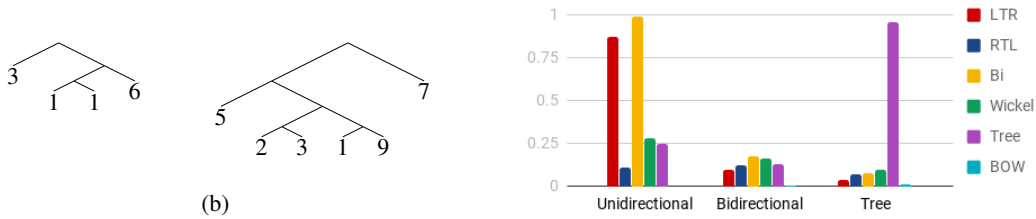
Role schemes: We consider 6 possible methods that networks might use to represent the roles of specific digits within a sequence; see Figure 3a for examples of these role schemes.

1. **Left-to-right:** Each digit's role is its index in the sequence, counting from left to right.
2. **Right-to-left:** Each digit's role is its index in the sequence, counting from right to left.
3. **Bidirectional:** Each digit's role is an ordered pair containing its left-to-right index and its right-to-left index (cf. human representations of spelling, Fischer-Baum et al. 2010).
4. **Wickelroles:** Each digit's role is the digit before it and the digit after it (Wickelgren, 1969). The edges of the sequence are marked with the special symbol #. Such roles were used by Rumelhart et al. (1986) in modeling English past tense formation.
5. **Tree positions:** Each digit's role is its position in a tree, such as RRL (left child of right child of right child of root). The tree structures considered are the outputs of the digit-parsing algorithm in Appendix C.
6. **Bag-of-words:** All digits have the same role. We call this a *bag-of-words* because it represents which digits ("words") are present and in what quantities, but ignores their positions.

Hypothesis: We hypothesize that RNN autoencoders will learn to use role representations that parallel their architectures: left-to-right roles for a unidirectional network, bidirectional roles for a bidirectional network, and tree-position roles for a tree-based network.

	3	1	1	6	5	2	3	1	9	7
Left-to-right	0	1	2	3	0	1	2	3	4	5
Right-to-left	3	2	1	0	5	4	3	2	1	0
Bidirectional	(0, 3)	(1, 2)	(2, 1)	(3, 0)	(0, 5)	(1, 4)	(2, 3)	(3, 2)	(4, 1)	(5, 0)
Wickelroles	#_1	3_1	1_6	1_#	#_2	5_3	2_1	3_9	1_7	9_#
Tree	L	RLL	RLR	RR	LL	LRLL	LRLR	LRRL	LRRR	R
Bag of words	r_0	r_0	r_0	r_0	r_0	r_0	r_0	r_0	r_0	r_0

(a)



(b) (c)

Figure 3: (a) The filler-role bindings assigned by the six role schemes to two sequences, 3116 and 523197. Roles not shown are assigned the null filler. (b) The trees used to assign tree roles to these sequences. (c) Substitution accuracy for three architectures at the autoencoding task with six role schemes. Each bar represents an average across five random initializations.

Evaluation: We evaluate how well a given sequence-to-sequence network can be approximated by a TPR with a particular role scheme as follows. First, we train a TPDN with the role scheme in question (Section 1.1). Then, we take the original encoder/decoder network and substitute the fitted TPDN for its encoder (Figure 2d). We do not conduct any additional training upon this substituted network; the decoder retains exactly the weights it learned in association with the original encoder, while the TPDN retains exactly the weights it learned for approximating the original encoder (including the weights on the final linear layer). We then compute the accuracy of the resulting hybrid network; we call this metric the **substitution accuracy**. High substitution accuracy indicates that the TPDN has approximated the encoder well enough for the decoder to handle the resulting vectors.

2.2 Results

Performance of seq2seq networks: The unidirectional and tree-based architectures both performed the training task nearly perfectly, with accuracies of 0.999 and 0.989 (averaged across five runs). Accuracy was lower (0.834) for the bidirectional architecture; this might mean that the hidden size of 60 becomes too small when divided into two 30-dimensional halves, one half for each direction.

Quality of TPDN approximation: For each of the six role schemes, we fitted a tensor product network onto the vectors generated by the trained encoder, and evaluated it using the substitution accuracy metric described in Section 2.1. The results are shown in Figure 3c. The different architectures do use different representations to solve the task. The tree-based autoencoder can be successfully approximated using tree-position roles and not using any of the other role schemes. By contrast, the unidirectional architecture is approximated very closely (with a substitution accuracy of over 0.99 averaged across five runs) by bidirectional roles. Left-to-right roles are also fairly successful (accuracy = 0.87), and right-to-left roles are decidedly unsuccessful (accuracy = 0.11). This asymmetry suggests that the unidirectional network is using *mildly bidirectional* roles: while it is best approximated by bidirectional roles, the representation strongly favors one direction over the other. Though the model uses the bidirectional role scheme, then, roles with the same left-to-right position (e.g. (2,3), (2,4), and (2,5)) can be collapsed without much loss of accuracy.

Finally, the bidirectional architecture is not approximated well by any of the role schemes we investigated. This could mean that the bidirectional architecture is implementing some other role scheme we did not consider. Alternately, it might have simply not adopted a robust method for representing sequence structure; this could explain why its accuracy on the training task was relatively low (0.83).

3 SENTENCE ENCODINGS

Will the success of TPDNs in approximating representations from RNN autoencoders of digit sequences extend to larger-scale representation? We explore this question using sentence representations extracted from four existing trained models: InferSent (Conneau et al., 2017), a bidirectional LSTM trained on the Stanford Natural Language Inference (SNLI) corpus (Bowman et al., 2015); Skip-thought (Kiros et al., 2015), an LSTM trained to predict, given a sentence, what the preceding or following sentence is; the Stanford sentiment model (SST) (Socher et al., 2013), a tree-based recursive neural tensor network trained to predict the sentiment of movie reviews; and SPINN (Bowman et al., 2016), a tree-based RNN trained on SNLI. More model details are in Appendix E.

3.1 TPDN APPROXIMATION

We now fit TPDNs to these four sentence encoding models. We experiment with all of the role schemes used in Section 2 except for Wickelroles; for sentence representations, the vocabulary size $|V|$ is so large that the Wickelrole scheme, which requires $|V|^2$ distinct roles, becomes intractable.

Preliminary experiments showed that the TPDN performed extremely poorly when learning the filler embeddings from scratch. Instead, we used pretrained word embeddings; for each model, we use the word embeddings used by that model. To partially fine-tune the embeddings, we train a linear transformation on top of the word embedding layer (though the embeddings themselves remain fixed). Thus, in this case, what the model has to learn are: the role embeddings, the linear transformation to apply to the fixed filler embeddings, and the final linear transformation applied to the sum of the filler/role bindings.

We train the decomposition on the sentence embeddings that each of the four models generate for all the premise sentences in the SNLI corpus (Bowman et al., 2015). We also tried instead using the sentences in the WikiText-2 corpus (Merity et al., 2016) but found better performance with the SNLI sentences. This is plausibly because the shorter, simpler sentences in the SNLI corpus made it easier for the model to learn the role embeddings without distraction from the fillers. For other training details see Appendix E. Table 1a shows the mean squared errors for various role schemes. In general, the mean squared errors show only small differences between the different role schemes, with perhaps the one exception being that tree-position roles do perform noticeably better than other role schemes for the SST model. Notably, bag-of-words roles perform nearly as well as the other role schemes, which is in stark contrast to the poor performance of bag-of-words roles in Section 2. Because mean squared error is not very interpretable on an absolute level, in the next section we perform more interpretable tests by looking at performance on downstream tasks.

3.2 PERFORMANCE ON DOWNSTREAM TASKS

Tasks: We assess how well the tensor product approximations perform at downstream tasks compared to the models they approximate. We use four tasks that are widely accepted for evaluations of sentence embeddings: (1) Stanford Sentiment Treebank (SST), rating the sentiment of movie reviews (Socher et al., 2013); (2) Microsoft Research Paraphrase Corpus (MRPC), classifying whether two sentences are paraphrases of each other (Dolan et al., 2004); (3) Semantic Textual Similarity Benchmark (STS-B), labeling how similar two sentences are on a continuous scale from 0 to 5 (Cer

(a)

	LTR	RTL	Bi	Tree	BOW
InferSent	0.17	0.18	0.17	**0.16**	0.19
Skip-thought	0.45	0.46	0.47	**0.42**	0.45
SST	0.24	0.26	0.26	**0.17**	0.27
SPINN	0.22	0.23	0.21	**0.18**	0.25

(b)

	LTR	RTL	Bi	Tree	BOW
InferSent	0.35	0.34	**0.29**	0.35	0.40
Skip-thought	0.34	0.37	**0.24**	0.34	0.51
SST	0.27	0.32	**0.25**	0.26	0.34
SPINN	0.49	0.53	**0.44**	0.49	0.56

Table 1: (a) Mean squared errors of TPDN approximations of sentence encodings (normalized by dividing by the mean squared error from training the TPDN on random vectors). (b) Performance of the sentence encoding models on our role-diagnostic analogies. Numbers indicate Euclidean distances (normalized by dividing by the average distance between vectors in the analogy set). Each column contains the average results for all analogies diagnostic of the role heading that column.

et al., 2017); and (4) Stanford Natural Language Inference (SNLI), determining whether one sentence entails a second sentence, contradicts the second sentence, or neither (Bowman et al., 2015).

Evaluation: We used SentEval (Conneau & Kiela, 2018) to train a classifier for each of the four tasks on the original model being approximated. We freeze this classifier and use it to classify the vectors generated by the TPDN. We then measure what proportion of the classifier's predictions for the approximation match its predictions for the original model. (We also try training SentEval classifiers on top of the TPDN instead of the original model; see Appendix F for these results. In general, for all models besides Skip-thought, the TPDN approximations perform nearly as well as the original models, and in some cases the approximations even outperform the originals.)

Results: For all tasks besides SNLI, we found that there was not a marked difference between bag-of-words roles and other role schemes. The results averaged across these tasks are in Table 2a. For SNLI, we did see instances where other role schemes outperformed bag-of-words (Table 2b).

Tables 2a and 2b both show that Skip-thought's behavior cannot be approximated very well by any of the role schemes that we considered. Setting aside Skip-thought, we now discuss the other three models. If these models truly contained no representation of structure, then bag-of-words roles would perform perfectly in Tables 2a and 2b. They fact that bag-of-words roles do not achieve accuracies at or very close to 1.0 indicates that there is some influence of structure on the models, and Table 2b (along with Table 1a) indicates that the types of structure that are weakly encoded appear to be tree-based for the tree-based models (SST and SPINN) and either tree-based or linear-index-based for InferSent. However, we can tell that this structural information is not represented so fully as to constitute a proper tensor product representation, because if it did we would expect the associated metrics in Tables 2a and 2b to be near 1.0 for the relevant structural role scheme, and to be near 0.0 for the bag-of-words roles, as was the case with the experiments in Section 2. Thus, overall we can conclude that the sentence encoding models do represent some weak notion of structure but that this representation of structure is not very robust or systematic and is largely overwhelmed by the fact that most of their behavior can be explained purely with a bag-of-words model.

3.3 ANALOGIES

Perhaps a more targeted approach can more clearly illuminate the presence of structure in these models. We therefore complement the TPDN analyses by examining analogies as a source of evidence for a model's representations. We construct sentence-based analogies that we would expect to hold only under certain role schemes. For example, consider the following analogy (expressed as an equation as in Mikolov et al. 2013):

$$\textbf{I see now} - \textbf{I see} = \textbf{you know now} - \textbf{you know} \tag{1}$$

A left-to-right role scheme makes (1) equivalent to (2) (f:r denotes the binding of filler f to role r):

$$\text{(I:0 + see:1 + now:2) – (I:0 + see:1) = (you:0 + know:1 + now:2) – (you:0 + know:1)} \tag{2}$$

In (2), both sides reduce to now:2, so (1) holds for representations using left-to-right roles. However, if (2) instead used right-to-left roles, it would not reduce in any clean way, so (1) would not hold.

(a)

Model	LTR	RTL	Bi	Tree	BOW
InferSent	**0.79**	**0.79**	0.78	0.78	0.77
Skip-thought	0.53	0.52	0.46	0.50	**0.58**
SST	**0.83**	0.82	0.82	0.82	0.81
SPINN	0.73	0.75	0.75	**0.76**	0.74

(b)

Model	LTR	RTL	Bi	Tree	BOW
InferSent	**0.77**	0.74	**0.77**	**0.77**	0.71
Skip-thought	**0.37**	**0.37**	0.36	0.36	**0.37**
SST	0.48	0.51	0.49	**0.67**	0.49
SPINN	0.72	0.72	0.73	**0.76**	0.58

Table 2: The proportion of test examples on which a classifier trained on a model gave the same prediction for the original model and for its TPDN approximation. (a) shows the average of these proportions across SST, MRPC, and STS-B, while (b) shows only SNLI. (For including STS-B results in the average in (a), we use the Pearson correlation between the original predictions and the TPDN predictions, and we add 1 to the correlation then divide by 2 to shift it to the same range as the accuracies).

We construct a dataset of role-diagnostic analogies, where each analogy should only hold for certain role schemes. For example, (1) works for left-to-right roles or bag-of-words roles, but not the other role schemes. The analogies use a vocabulary based on Ettinger et al. (2018) to ensure plausibility of the constructed sentences. For each analogy, we create 4 equations, one isolating each of the four terms (e.g. **I see = I see now – you know now + you know**). We then compute the Euclidean distance between the two sides of each equation using each model's encodings of the relevant sentences.

The results are in Table 1b. InferSent, Skip-thought, and SPINN all show results most consistent with bidirectional roles, while SST shows results most consistent with either tree-based or bidirectional roles. Notably, the bag-of-words column shows poor performance by all models, indicating that in controlled enough settings these models can be shown to have some more structured behavior even though evaluation on examples from applied tasks does not clearly bring out that structure. These analogies thus provide independent evidence for the conclusions of the TPDN analysis: these models have a weak notion of structure built into them, but that structure is largely drowned out by the non-structure-sensitive, bag-of-words aspects of their representations.

4 When do RNNs learn compositional representations?

The previous section found that all of the popular sentence representation architectures surveyed did not robustly encode structure and could even be approximated relatively well with a structureless bag-of-words model. Motivated by this finding, we now investigate how different aspects of training influence models' learned representations to gain insight into how models can be influenced to learn better compositionality. To conduct this investigation in a clearer and more interpretable way, we return to the setting (from Section 2) of operating over sequences of digits. We investigate two aspects of training: the architecture and the training task.

Teasing apart the contribution of the encoder and decoder: In Section 2, we investigated autoencoders whose encoder and decoder had the same topology (whether unidirectional, bidirectional, or tree-based). To understand the extent to which each of the two components of the network contributes to the type of representation that is learned, we now expand the investigation to include networks where the encoder and decoder differ. We find that the decoder largely dictates what type of roles are learned: architectures with unidirectional decoders prefer mildly bidirectional roles, architectures with bidirectional decoders fail to be well-approximated by any role scheme, and architectures with tree-based decoders are best approximated by tree-based roles. However, there is still some effect of the encoder; most noticeably, in the tree/uni and tree/bi architectures, the tree-position roles perform better than they do for the other architectures with the same decoders. (For the exact results, see Appendix D.) Though work on novel architectures often focuses on proposing novel encoders, this finding suggests that it may be more important to focus on the decoder for getting neural networks to learn representations structured in a certain way.

The contribution of the training task: We next explore how the training task affects the representations that are learned. We test four tasks, illustrated in Table 3a: **autoencoding** (returning the input sequence unchanged), **reversal** (reversing the input), **sorting** (returning the input digits in ascending order), and **interleaving** (alternating digits from the left and right edges of the input).

Table 3b gives the substitution accuracy for a TPDN trained to approximate a unidirectional encoder that was trained with a unidirectional decoder on each task. We observe that training task does play a noticeable role in determining the learned representations. First, though the model has learned mildly bidirectional roles favoring the left-to-right direction for autoencoding, for reversal the right-to-left direction is far preferred over left-to-right (indeed, it is so strong that the reversal model is probably most properly said to be using right-to-left roles, not bidirectional ones). For the interleaving task, the model is approximated best with strongly bidirectional roles: that is, bidirectional roles work nearly perfectly, while neither unidirectional scheme works well. Finally, on the sorting task (in stark contrast to the other tasks), bag-of-words roles work nearly as well as all the other schemes, suggesting that the model has learned to discard most structural information given that the sorting task does not require structural information. These experiments therefore show how the task can either encourage or discourage the model from developing representations of structure.

(a)

Input	34097	436513
Autoencode	34097	436513
Reverse	79043	315634
Sort	03479	133456
Interleave	37490	433165

(b)

	LTR	RTL	Bi	Wickel	Tree	BOW
Autoencode	0.87	0.11	**0.99**	0.28	0.25	0.00
Reverse	0.06	0.99	**1.00**	0.18	0.20	0.00
Sort	0.90	0.90	**0.92**	0.88	0.89	0.89
Interleave	0.27	0.18	**0.99**	0.63	0.36	0.00

Table 3: (a) Tasks used to test for the effect of task on learned roles (Section 4). (b) Accuracy of the TPDN applied to models trained on these tasks with a unidirectional encoder and decoder. All numbers are averages across five random initializations.

5 Related work

There are several approaches for interpreting the behavior of neural networks. One approach infers the information encoded in the representations from the behavior of the system on examples that measure specific dimensions of the generalization, for example in semantics (Pavlick, 2017; Dasgupta et al., 2018; Poliak et al., 2018) or syntax (Linzen et al., 2016). Another approach is based on probing tasks, which assess what information can be easily decoded from a vector representation (Shi et al. 2016; Belinkov et al. 2017; Kádár et al. 2017; Ettinger et al. 2018; cf. work in cognitive neuroscience, e.g. Norman et al. 2006). Our method is wider-reaching than the probing task approach, or the Mikolov et al. (2013) analogy approach: instead of decoding a single feature, we attempt to exhaustively decompose the vector space into a linear combination of filler-role bindings.

The TPDN's successful decomposition of sequence representations in our experiments shows that RNNs can sometimes be approximated with no nonlinearities or recurrence. This finding is related to Levy et al. (2018), who argued that LSTMs dynamically compute weighted sums of their inputs; TPRs replace the weights of the sum with the role vectors. Levy et al. (2018) also showed that recurrence is largely unnecessary for practical applications. Vaswani et al. (2017) report very good performance for a sequence model without recurrence; importantly, they find it necessary to incorporate sequence position embeddings, which are similar to the left-to-right roles discussed in Section 2. Methods for interpreting neural networks using more interpretable architectures have been proposed before based on rules and automata (Omlin & Giles, 1996; Weiss et al., 2018). Finally, there is a large body of work on incorporating explicit symbolic representations into neural networks (for a recent review, see Battaglia et al. 2018); indeed, tree-shaped RNNs are an example of this approach. While our work is orthogonal to this line of work, we note that TPRs can be profitably used as an explicit component of neural networks (Palangi et al., 2018).

6 Conclusion

What kind of internal representations could allow simple sequence-to-sequence models to perform the remarkable feats they do, including tasks previously thought to require compositional, symbolic representations (e.g., translation)? Perhaps, for structure-sensitive tasks, these models implicitly learn tensor-product representations (TPRs), which are distributed embeddings of symbol structures that enable powerful symbolic computation to be performed with neural operations. This seems a priori unlikely, since TPRs (and related methods) have simple, multi-linear structure, and high-performing sequence-to-sequence architectures have recurrence, non-linearities, and multiplicative gating. Yet the work reported here shows that in heavily structure-sensitive tasks, sequence-to-sequence models do learn representations that are extremely well approximated by TPRs; this was demonstrated with networks performing pure sequence-manipulation tasks. This method of approximating learned internal representations via TPRs is carried out by the proposed tensor-product decomposition network (TPDN). TPDNs enable the study of how network architecture and task variations induce different types and degrees of structure-sensitivity in representations; they indicate, for example, that the decoder plays a greater role than the encoder in determining the structure of the learned representation. TPDNs applied to mainstream sentence-embedding models reveal that unstructured bag-of-words models provide a respectable approximation; nonetheless, this experiment does also provide evidence for a moderate degree of structure-sensitivity. The presence of structure-sensitivity is corroborated by targeted analogy tests motivated by the linearity of TPRs. Overall, TPDNs provide an illuminating tool for understanding structure in learned vector representations.

REFERENCES

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. What do neural machine translation models learn about morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 861–872, Vancouver, Canada, 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-1080.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1075.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1466–1477. Association for Computational Linguistics, 2016. URL http://aclweb.org/anthology/P16-1139.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14. Association for Computational Linguistics, 2017. doi: 10.18653/v1/S17-2001. URL http://www.aclweb.org/anthology/S17-2001.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1936–1945. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1177. URL http://www.aclweb.org/anthology/P17-1177.

Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. *arXiv preprint arXiv:1802.03691*, 2018.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

Alexis Conneau and Douwe Kiela. Senteval: An evaluation toolkit for universal sentence representations. In *International Conference on Language Resources and Evaluation*, 2018.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 670–680. Association for Computational Linguistics, 2017. URL http://aclweb.org/anthology/D17-1070.

Ishita Dasgupta, Demi Guo, Andreas Stuhlmüller, Samuel J. Gershman, and Noah D. Goodman. Evaluating compositionality in sentence embeddings. In *Proceedings of the 40th Annual Conference of the Cognitive Science Society*, 2018. URL https://arxiv.org/abs/1802.04302.

Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, 2004. URL http://www.aclweb.org/anthology/C04-1051.

Allyson Ettinger, Ahmed Elgohary, Colin Phillips, and Philip Resnik. Assessing composition in sentence vector representations. In *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1790–1801. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/C18-1152.

Simon Fischer-Baum, Michael McCloskey, and Brenda Rapp. Representation of letter position in spelling: Evidence from acquired dysgraphia. *Cognition*, 115(3):466–490, 2010.

Ákos Kádár, Grzegorz Chrupała, and Afra Alishahi. Representation of linguistic form and function in recurrent neural networks. *Computational Linguistics*, 43(4):761–780, 2017. URL http://aclweb.org/anthology/J17-4003.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in Neural Information Processing Systems*, pp. 3294–3302, 2015.

Omer Levy, Kenton Lee, Nicholas FitzGerald, and Luke Zettlemoyer. Long short-term memory as a dynamically computed element-wise weighted sum. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 732–739. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/P18-2116.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, pp. 746–751, 2013.

Kenneth A. Norman, Sean M. Polyn, Greg J. Detre, and James V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fMRI data. *Trends in Cognitive Sciences*, 10(9):424–430, 2006.

Christian W. Omlin and C. Lee Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

Hamid Palangi, Paul Smolensky, Xiaodong He, and Li Deng. Question-answering with grammatically-interpretable representations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17090.

Ellie Pavlick. *Compositional lexical semantics in natural language inference*. PhD thesis, University of Pennsylvania, 2017.

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

Tony A. Plate. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3): 623–641, 1995.

Adam Poliak, Yonatan Belinkov, James Glass, and Benjamin Van Durme. On the evaluation of semantic phenomena in neural machine translation using natural language inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 513–523. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-2082.

Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105, 1990.

David E. Rumelhart, James L. McClelland, and PDP Research Group. *Parallel distributed processing*, volume 1. MIT Press, Cambridge, MA, 1986.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1526–1534. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1159. URL `http://www.aclweb.org/anthology/D16-1159`.

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

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, 2013.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-1150`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *ICML*, pp. 5244–5253, 2018.

Wayne A. Wickelgren. Context-sensitive coding, associative memory, and serial order in (speech) behavior. *Psychological Review*, 76(1):1–15, 1969.

A LIST OF ACRONYMS AND ABBREVIATIONS

Bi	Bidirectional
BOW	Bag of words
LTR	Left to right
MRPC	Microsoft Research Paraphrase Corpus
RTL	Right to left
SNLI	Stanford Natural Language Inference corpus
SST	Stanford Sentiment Treebank
STS-B	Semantic Textual Similarity Benchmark
TPDN	Tensor product decomposition network
TPR	Tensor product representation
Uni	Unidirectional
Wickel	Wickelroles (see Section 2.1)

B ANALYSIS OF ARCHITECTURE COMPONENTS

Here we analyze how several aspects of the TPDN architecture contribute to our results. For all of the experiements described in this section, we used TPDNs to approximate a sequence-to-sequence network with a unidirectional encoder and unidirectional decoder that was trained to perform the reversal task (Section 4); we chose this network because it was strongly approximated by right-to-left roles, which are relatively simple (but still non-trivial).

B.1 IS THE FINAL LINEAR LAYER NECESSARY?

One area where our model diverges from traditional tensor product representations is in the presence of the final linear layer (step 5 in Figure 2c). This layer is necessary if one wishes to have freedom to choose the dimensionality of the filler and role embeddings; without it, the dimensionality of the representations that are being approximated must factor exactly into the product of the dimensionality of the filler embeddings and the dimensionality of the role embedding (see Figure 2c). It is natural to wonder whether the only contribution of this layer is in adjusting the dimensionality or whether it serves a broader function. Table 4 shows the results of approximating the reversal sequence-to-sequence network with and without this layer; it indicates that this layer is highly necessary for the successful decomposition of learned representations.

B.2 VARYING THE DIMENSIONALITY OF THE FILLER AND ROLE EMBEDDINGS

Two of the parameters that must be provided to the TPDN are the dimensionality of the filler embeddings and the dimensionality of the role embeddings. We explore the effects of these parameters in Figure 4. We find that, for the role embeddings, substitution accuracy increases noticeably with each increase in dimensionality until the dimensionality hits 6, where accuracy plateaus. This behavior is likely due to the fact that the reversal seq2seq network is most likely to employ right-to-left roles, which involves 6 possible roles in this setting. A dimensionality of 6 is therefore the minimum embedding size needed to make the role vectors linearly independent; linear independence is an important property for the fidelity of a tensor product representation (Smolensky, 1990). The accuracy also generally increases as filler dimensionality increases, but there is a less clear point where it plateaus for the fillers than for the roles.

B.3 FILLER-ROLE BINDING OPERATION

The body of the paper focused on using the tensor product ($f_i \otimes r_i$, see Figure 2b) as the operation for binding fillers to roles. There are other conceivable binding operations. Here we test two alternatives, both of which can be viewed as special cases of the tensor product or as related to it: circular convolution, which is used in holographic reduced representations (Plate, 1995), and elementwise product ($f_i \odot r_i$). Both of these are restricted such that roles and fillers must have the same embedding dimension ($N_f = N_r$). We first try setting this dimension to 20, which is what was used as both the role and filler dimension in all tensor product experiments with digit sequences.

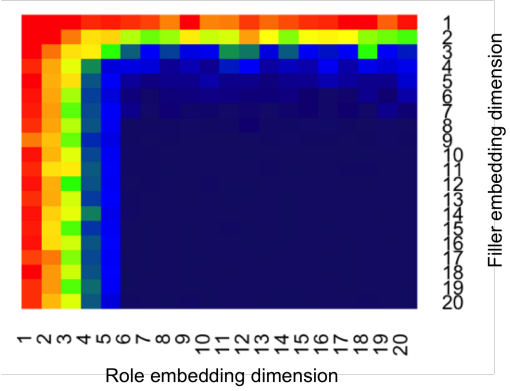


Figure 4: Heatmap of substitution accuracies with various filler and role embedding dimensionalities. Red indicates accuracy under 1%; dark blue indicates accuracy over 80%. The models whose substitution accuracies are displayed are all TPDNs trained to approximate a sequence-to-sequence model that was trained on the task of reversal.

We found that while these dimensions were effective for the tensor product binding operation, they were not effective for elementwise product and circular convolution (Table 5). When the dimension was increased to 60, however, the elementwise product performed roughly as well as as the tensor product; circular convolution now learned one of the two viable role schemes (right-to-left roles) but failed to learn the equally viable bidirectional role scheme. Thus, our preliminary experiments suggest that these other two binding operations do show promise, but seem to require larger embedding dimensions than tensor products do. At the same time, they still have fewer parameters overall compared to the tensor product because their final linear layers (of dimensionality N) are much smaller than those used with a tensor product (of dimensionality N^2).

C THE DIGIT PARSING ALGORITHM

When inputting digit sequences to our tree-based model, the model requires a predefined tree structure for the digit sequence. We use the following algorithm to generate this tree structure: at each timestep, combine the smallest element of the sequence (other than the last element) with its neighbor immediately to the right, and replace the pair with that neighbor. If there is a tie for the smallest digit, choose the leftmost tied digit.

For example, the following shows step-by-step how the tree for the sequence 523719 would be generated:

- 5 2 3 7 1 9
- 5 2 3 7 [1 9]
- 5 [2 3] 7 [1 9]
- 5 [[2 3] 7] [1 9]
- [5 [[2 3] 7] [1 9]]

D FULL RESULTS OF SEQUENCE-TO-SEQUENCE EXPERIMENTS

Section 4 summarized the results of our experiments which factorially varied the training task, the encoder and the decoder. Here we report the full results of these experiments in two tables: Table 6 shows the accuracies achieved by the sequence-to-sequence models at the various training tasks, and Table 7 shows the substitution accuracies of TPDNs applied to the trained sequence-to-sequence models for all architectures and tasks.

E MODEL AND TRAINING DETAILS

E.1 SEQUENCE-TO-SEQUENCE MODELS

As much as possible, we standardized parameters across all sequence-to-sequence models that we trained on the digit-sequence tasks.

For all decoders, when computing a new hidden state, the only input to the recurrent unit is the previous hidden state (or parent hidden state, for a tree-based decoder), without using any previous outputs as inputs to the hidden state update. This property is necessary for using a bidirectional decoder, since it would not be possible to generate the output both before and after each bidirectional decoder hidden state.

We also inform the decoder of when to stop decoding; that is, for sequential models, the decoder stops once its output is the length of the sequence, while for tree-based models we tell the model which positions in the tree are leaves. Stopping could alternately be determined by some action of the decoder (e.g., generating an end-of-sequence symbol); for simplicity we chose the strategy outlined above instead.

For all architectures, we used a digit embedding dimensionality of 10 (chosen arbitrarily) and a hidden layer size of 60 (this hidden layer size was chosen because 60 has many integer factors, making it amenable to the dimensionality analyses in Appendix B.2). For the bidirectional architectures, the forward and backward recurrent layers each had a hidden layer size of 30, so that their concatenated hidden layer size was 60. For bidirectional decoders, a linear layer condensed the 60-dimensional encoding into 30 dimensions before it was passed to the forward and backward decoders.

The networks were trained using the Adam optimizer (Kingma & Ba, 2015) with the standard initial learning rate of 0.001. We used negative log likelihood, computed over the softmax probability distributions for each output sequence element, as the loss function. Training proceeded with a batch size of 32, with loss on the held out development set computed after every 1,000 training examples. Training was halted when the loss on the heldout development set had not improved for any of the development loss checkpoints for a full epoch of training (i.e. 40,000 training examples). Once training completed, the parameters from the best-performing checkpoint were reloaded and used for evaluation of the network.

E.2 TPDNS TRAINED ON DIGIT MODELS

When applying TPDNs to the digit-based sequence-to-sequence models, we always used 20 as both the filler embedding dimensionality and role embedding dimensionality. This decision was based on the experiments in Appendix B.2; we selected filler and role embedding dimensionalities that were safely above the cutoff needed to lead to successful decomposition.

The TPDNs were trained with the same training regimen as the sequence-to-sequence models, except that, instead of using negative log likelihood as the loss function, for the TPDNs we used mean squared error between the predicted vector representation and the actual vector representation from the original sequence-to-sequence network.

E.3 SENTENCE EMBEDDING MODELS

For all four sentence encoding models, we used publicly available and freely downloadable pre-trained versions found at the following links:

- InferSent: `https://github.com/facebookresearch/InferSent`
- Skip-thought: `https://github.com/ryankiros/skip-thoughts`
- SST: `https://nlp.stanford.edu/software/corenlp.shtml`
- SPINN: `https://github.com/stanfordnlp/spinn`

InferSent is a bidirectional LSTM with 4096-dimensional hidden states. For Skip-thought, we use the unidirectional variant, which is an LSTM with 2400-dimensional hidden states. The SST model is a recurrent neural tensor network (RNTN) with 25-dimensional hidden states. Finally, for SPINN,

we use the SPINN-PI-NT version, which is equivalent to a tree-LSTM (Tai et al., 2015) with 300-dimensional hidden states.

E.4 TPDNs TRAINED ON SENTENCE MODELS

For training a TPDN to approximate the sentence encoding models, the filler embedding dimensions were dictated by the size of the pretrained word embeddings; these dimensions were 300 for InferSent and SPINN, 620 for Skip-thought, and 25 for SST. The linear transformation applied to the word embeddings did not change their size. For role embedding dimensionality we tested all role dimensions in $\{1, 5, 10, 20, 40, 60\}$. The best-performing dimension was chosen based on preliminary experiments and used for all subsequent experiments; we thereby chose role dimensionalities of 10 for InferSent and Skip-thought, 20 for SST, and 5 for SPINN. In general, role embedding dimensionalities of 5, 10, and 20 all performed noticeably better than 1, 40, and 60, but there was not much difference between 5, 10, and 20.

The training regimen for the TPDNs on sentence models was the same as for the TPDNs trained on digit sequences.

F DOWNSTREAM TASK PERFORMANCE FOR TPDNs APPROXIMATING SENTENCE ENCODERS

For each TPDN trained to approximate a sentence encoder, we evaluate it on four downstream tasks: (i) Stanford Sentiment Treebank (SST), which is labeling the sentiment of movie reviews (Socher et al., 2013); this task is further subdivided into SST2 (labeling the reviews as *positive* or *negative*) and SST5 (labeling the reviews on a 5-point scale, where 1 means *very negative* and 5 means *very positive*). The metric we report for both tasks is accuracy. (ii) Microsoft Research Paraphrase Corpus (MRPC), which is labeling whether two sentences are paraphrases of each other (Dolan et al., 2004). For this task, we report both accuracy and F1. (iii) Semantic Textual Similarity Benchmark (STS-B), which is giving a pair of sentences a score on a scale from 0 to 5 indicating how similar the two sentences are (Cer et al., 2017). For this task, we report Pearson and Spearman correlation coefficients. (iv) Stanford Natural Language Inference (SNLI), which involves labeling two sentences to indicate whether the first entails the second, contradicts the second, or neither (Bowman et al., 2015). For this task, we report accuracy as the evaluation metric.

F.1 SUBSTITUTION PERFORMANCE

The first results we report for the TPDN approximations of sentence encoders is similar to the substitution accuracy used for digit encoders. Here, we use SentEval (Conneau & Kiela, 2018) to train linear classifiers for all downstream tasks on the original sentence encoding model; then, we freeze the weights of these classifiers and use them to classify the test-set encodings generated by the TPDN approximation. We use the classifier parameters recommended by the SentEval authors: using a linear classifier (not a multi-layer perceptron) trained with the Adam algorithm (Kingma & Ba, 2015) using a batch size of 64, a tenacity of 5, and an epoch size of 4. The results are shown in Table 8.

F.2 AGREEMENT BETWEEN THE TPDN AND THE ORIGINAL MODEL

Next, we analyze the same results from the previous section, but instead of reporting accuracies we report the extent to which the TPDN's predictions agree with the original model's predictions. For SST, MRPC, and SNLI, this agreement is defined as the proportion of their labels that are the same. For STS-B, the agreement is the Pearson correlation between the original model's outputs and the TPDN's outputs. The results are in Table 9.

F.3 TRAINING A CLASSIFIER ON THE TPDN

Finally, we consider treating the TPDNs as models in their own right and use SentEval to both train and test downstream task classifiers on the TPDNs. The results are in Table 10.

Filler dim.	Role dim.	Without linear layer	With linear layer
1	60	0.0002	0.003
2	30	0	0.042
3	20	0.001	0.82
4	15	0.0006	0.80
5	12	0	0.90
6	10	0	0.92
10	6	0.0002	0.99
12	5	0	0.67
15	4	0.0002	0.37
20	3	0	0.14
30	2	0.0002	0.02
60	1	0.001	0.0014

Table 4: Substitution accuracies with and without the final linear layer, for TPDNs using various combinations of filler and role embedding dimensionality. These TPDNs were approximating a seq2seq model trained to perform reversal.

	LTR	RTL	Bi	Wickel	Tree	BOW	Parameters
Tensor product (20 dim.)	0.054	0.993	0.996	0.175	0.188	0.002	24k
Tensor product (60 dim.)	0.046	0.988	0.996	0.138	0.172	0.001	217k
Circular convolution (20 dim.)	0.004	0.045	0.000	0.000	0.003	0.001	1.5k
Circular convolution (60 dim.)	0.048	0.964	0.066	0.001	0.013	0.001	4.6k
Elementwise product (20 dim.)	0.026	0.617	0.386	0.024	0.027	0.001	1.5k
Elementwise product (60 dim.)	0.051	0.992	0.993	0.120	0.173	0.001	4.6k

Table 5: Approximating a unidirectional seq2seq model trained to perform sequence reversal: substitution accuracies using different binding operations.

Encoder	Decoder	Autoencode	Reverse	Sort	Interleave
Uni	Uni	0.999	1.000	1.000	1.000
Uni	Bi	0.949	0.933	1.000	0.968
Uni	Tree	0.979	0.967	0.970	0.964
Bi	Uni	0.993	0.999	1.000	0.995
Bi	Bi	0.834	0.883	1.000	0.939
Bi	Tree	0.967	0.920	0.959	0.909
Tree	Uni	0.981	0.978	1.000	0.987
Tree	Bi	0.891	0.900	1.000	0.894
Tree	Tree	0.989	0.962	0.999	0.934

Table 6: Accuracies of the various sequence-to-sequence encoder/decoder combinations at the different training tasks. Each number in this table is an average across five random initializations. Uni = unidirectional; bi = bidirectional.

Task	Encoder	Decoder	LTR	RTL	Bi	Wickel	Tree	BOW
Autoencode	Uni	Uni	0.871	0.112	**0.992**	0.279	0.246	0.001
	Uni	Bi	0.275	0.273	**0.400**	0.400	0.238	0.005
	Uni	Tree	0.053	0.086	0.094	0.105	**0.881**	0.009
	Bi	Uni	0.748	0.136	**0.921**	0.209	0.207	0.002
	Bi	Bi	0.097	0.124	**0.179**	0.166	0.128	0.006
	Bi	Tree	0.051	0.081	0.088	0.095	**0.835**	0.007
	Tree	Uni	0.708	0.330	**0.865**	0.595	0.529	0.007
	Tree	Bi	0.359	0.375	0.491	**0.569**	0.376	0.009
	Tree	Tree	0.041	0.069	0.076	0.095	**0.958**	0.009
Reverse	Uni	Uni	0.062	0.986	**0.995**	0.177	0.204	0.002
	Uni	Bi	0.262	0.268	0.386	**0.406**	0.228	0.006
	Uni	Tree	0.103	0.112	0.177	0.169	**0.413**	0.002
	Bi	Uni	0.037	0.951	**0.965**	0.084	0.146	0.001
	Bi	Bi	0.121	0.140	**0.228**	0.170	0.140	0.005
	Bi	Tree	0.085	0.105	0.17	0.151	**0.385**	0.002
	Tree	Uni	0.178	0.755	**0.802**	0.424	0.564	0.007
	Tree	Bi	0.302	0.332	0.442	**0.549**	0.368	0.009
	Tree	Tree	0.083	0.096	0.147	0.152	**0.612**	0.004
Sort	Uni	Uni	0.895	0.895	**0.923**	0.878	0.890	0.892
	Uni	Bi	0.898	0.894	**0.923**	0.916	0.915	0.904
	Uni	Tree	0.218	0.212	0.207	0.193	**0.838**	0.275
	Bi	Uni	0.886	0.884	**0.917**	0.812	0.871	0.847
	Bi	Bi	0.921	0.925	**0.945**	0.835	0.927	0.934
	Bi	Tree	0.219	0.216	0.209	0.194	**0.816**	0.273
	Tree	Uni	0.997	0.998	0.997	**0.999**	0.999	0.998
	Tree	Bi	**1.0**	1.000	0.997	1.000	1.000	1.000
	Tree	Tree	0.201	0.199	0.179	0.181	**0.978**	0.249
Interleave	Uni	Uni	0.269	0.181	**0.992**	0.628	0.357	0.003
	Uni	Bi	0.177	0.095	**0.728**	0.463	0.255	0.005
	Uni	Tree	0.040	0.033	0.116	0.089	**0.373**	0.003
	Bi	Uni	0.186	0.126	**0.965**	0.438	0.232	0.001
	Bi	Bi	0.008	0.074	**0.600**	0.128	0.162	0.002
	Bi	Tree	0.031	0.025	0.069	0.057	**0.395**	0.004
	Tree	Uni	0.330	0.208	**0.908**	0.663	0.522	0.005
	Tree	Bi	0.191	0.151	**0.643**	0.518	0.391	0.006
	Tree	Tree	0.027	0.025	0.059	0.069	**0.606**	0.004

Table 7: Substitution accuracies for TPDNs applied to all combinations of encoder, decoder, training task, and hypothesized role scheme. Each number is an average across five random initializations. Uni = unidirectional; bi = bidirectional.

	LTR	RTL	Bi	Tree	BOW	Original
InferSent						
SST2	0.79	0.79	0.77	0.79	0.80	0.85
SST5	0.40	0.39	0.40	0.41	0.42	0.46
MRPC (accuracy)	0.70	0.71	0.72	0.70	0.72	0.73
MRPC (F1)	0.81	0.82	0.82	0.80	0.81	0.81
STS-B (Pearson)	0.69	0.70	0.71	0.70	0.69	0.78
STS-B (Spearman)	0.68	0.68	0.69	0.68	0.67	0.78
SNLI	0.71	0.69	0.72	0.71	0.66	0.84
Skip-thought						
SST2	0.58	0.51	0.50	0.50	0.61	0.81
SST5	0.29	0.27	0.21	0.25	0.31	0.43
MRPC (accuracy)	0.60	0.62	0.62	0.61	0.66	0.74
MRPC (F1)	0.73	0.75	0.76	0.75	0.79	0.82
STS-B (Pearson)	-0.01	-0.07	-0.10	-0.05	0.06	0.73
STS-B (Spearman)	0.23	-0.01	-0.05	0.00	0.08	0.72
SNLI	0.35	0.35	0.34	0.35	0.35	0.73
SST						
SST2	0.76	0.76	0.76	0.75	0.77	0.83
SST5	0.37	0.38	0.37	0.37	0.38	0.45
MRPC (accuracy)	0.67	0.67	0.67	0.66	0.66	0.66
MRPC (F1)	0.80	0.80	0.80	0.79	0.80	0.80
STS-B (Pearson)	0.24	0.21	0.22	0.19	0.24	0.29
STS-B (Spearman)	0.24	0.22	0.23	0.20	0.25	0.27
SNLI	0.40	0.41	0.41	0.41	0.40	0.42
SPINN						
SST2	0.73	0.73	0.73	0.74	0.74	0.76
SST5	0.36	0.36	0.35	0.37	0.37	0.39
MRPC (accuracy)	0.67	0.68	0.67	0.67	0.68	0.70
MRPC (F1)	0.75	0.78	0.76	0.76	0.76	0.79
STS-B (Pearson)	0.60	0.60	0.62	0.62	0.53	0.67
STS-B (Spearman)	0.58	0.59	0.59	0.59	0.57	0.65
SNLI	0.67	0.67	0.68	0.69	0.54	0.79

Table 8: Substitution results on performing the applied tasks for TPDNs trained to approximate the representations from each of the four downloaded models. For MRPC, we report accuracy and F1. For STS-B, we report Pearson correlation and Spearman correlation. All other metrics are accuracies.

	LTR	RTL	Bi	Tree	BOW	Original
Infersent						
SST2	0.85	0.84	0.82	0.83	0.84	1.00
SST5	0.60	0.59	0.58	0.59	0.61	1.00
MRPC	0.78	0.80	0.78	0.77	0.79	1.00
STS-B	0.87	0.86	0.87	0.86	0.84	1.00
SNLI	0.77	0.74	0.77	0.77	0.71	1.00
Skip-thought						
SST2	0.58	0.54	0.50	0.51	0.60	1.00
SST5	0.41	0.41	0.18	0.35	0.43	1.00
MRPC	0.69	0.71	0.74	0.72	0.79	1.00
STS-B	-0.10	-0.12	-0.16	-0.13	-0.04	1.00
SNLI	0.37	0.37	0.36	0.36	0.37	1.00
SST						
SST2	0.84	0.84	0.83	0.84	0.85	1.00
SST5	0.65	0.64	0.64	0.64	0.65	1.00
MRPC	0.99	0.99	0.99	0.98	0.99	1.00
STS-B	0.64	0.59	0.62	0.68	0.60	1.00
SNLI	0.48	0.51	0.49	0.67	0.49	1.00
SPINN						
SST2	0.77	0.77	0.77	0.79	0.79	1.00
SST5	0.61	0.62	0.63	0.63	0.62	1.00
MRPC	0.68	0.73	0.72	0.74	0.70	1.00
STS-B	0.72	0.73	0.74	0.76	0.70	1.00
SNLI	0.72	0.72	0.73	0.76	0.58	1.00

Table 9: The proportion of times that a classifier trained on a sentence encoding model gave the same downstream-task predictions based on the original sentence encoding model and based on a TPDN approximating that model, where the TPDN uses the role schemes indicated by the column header. For all tasks but STS-B, these numbers show the proportion of predictions that matched; chance performance is 0.5 for SST2 and MRPC, 0.2 for SST5, and 0.33 for SNLI. For STS-B, the metric shown is the Pearson correlation between the TPDN's similarity ratings and the original model's similarity ratings; chance performance here is 0.0.

	LTR	RTL	Bi	Tree	BOW	Original
Infersent						
SST2	0.82	0.82	0.81	0.81	0.83	0.85
SST5	0.44	0.44	0.44	0.44	0.43	0.46
MRPC (acc.)	0.71	0.73	0.72	0.70	0.73	0.73
MRPC (F1)	0.80	0.81	0.81	0.80	0.81	0.81
STS-B (Pearson)	0.71	0.71	0.71	0.71	0.71	0.78
STS-B (Spearman)	0.69	0.70	0.70	0.69	0.70	0.78
SNLI	0.77	0.76	0.77	0.77	0.75	0.84
Skip-thought						
SST2	0.59	0.56	0.50	0.52	0.61	0.81
SST5	0.30	0.30	0.25	0.28	0.32	0.43
MRPC (acc.)	0.67	0.66	0.66	0.66	0.67	0.74
MRPC (F1)	0.80	0.80	0.80	0.80	0.79	0.82
STS-B (Pearson)	0.19	0.17	0.13	0.13	0.23	0.73
STS-B (Spearman)	0.17	0.16	0.08	0.10	0.20	0.72
SNLI	0.47	0.46	0.46	0.44	0.46	0.73
SST						
SST2	0.78	0.78	0.77	0.78	0.79	0.83
SST5	0.40	0.40	0.40	0.39	0.41	0.45
MRPC (acc.)	0.67	0.67	0.66	0.66	0.67	0.66
MRPC (F1)	0.80	0.80	0.80	0.80	0.80	0.88
STS-B (Pearson)	0.28	0.23	0.23	0.20	0.28	0.29
STS-B (Spearman)	0.27	0.23	0.23	0.20	0.27	0.27
SNLI	0.45	0.44	0.45	0.43	0.45	0.42
SPINN						
SST2	0.79	0.78	0.77	0.77	0.80	0.76
SST5	0.42	0.42	0.40	0.42	0.42	0.39
MRPC (acc.)	0.72	0.71	0.71	0.68	0.72	0.70
MRPC (F1)	0.81	0.80	0.80	0.79	0.80	0.79
STS-B (Pearson)	0.68	0.67	0.67	0.67	0.66	0.67
STS-B (Spearman)	0.67	0.66	0.66	0.65	0.65	0.65
SNLI	0.72	0.71	0.72	0.73	0.71	0.79

Table 10: Downstream task performance for classifiers trained and tested on the TPDNs that were trained to approximate each of the four applied models. The rightmost column indicates the performance of the original model (without the TPDN approximation).