

Correlating neural and symbolic representations of language

Grzegorz Chrupala

Tilburg University

g.chrupala@uvt.nl

Afra Alishahi

Tilburg University

a.alishahi@uvt.nl

Abstract

Analysis methods which enable us to better understand the representations and functioning of neural models of language are increasingly needed as deep learning becomes the dominant approach in NLP. Here we present two methods based on Representational Similarity Analysis (RSA) and Tree Kernels (TK) which allow us to directly quantify how strongly the information encoded in neural activation patterns corresponds to information represented by symbolic structures such as syntax trees. We first validate our methods on the case of a simple synthetic language for arithmetic expressions with clearly defined syntax and semantics, and show that they exhibit the expected pattern of results. We then apply our methods to correlate neural representations of English sentences with their constituency parse trees.

1 Introduction

Analysis methods which allow us to better understand the representations and functioning of neural models of language are increasingly needed as deep learning becomes the dominant approach to natural language processing. A popular technique for analyzing neural representations involves predicting information of interest from the activation patterns, typically using a simple predictive model such as a linear classifier or regressor. If the model is able to predict this information with high accuracy, the inference is that the neural representation encodes it. We refer to these as *diagnostic models*.

One important limitation of this method of analysis is that it is only easily applicable to relatively simple types of target information, which are amenable to be predicted via linear regression or classification. Should we wish to decode activation patterns into a structured target such as a syntax tree, we would need to resort to complex

structure prediction algorithms, running the risk that the analytic method becomes no simpler than the actual neural model.

Here we introduce an alternative approach based on correlating neural representations of sentences and structured symbolic representations commonly used in linguistics. Crucially, the correlation is in similarity space rather than in the original representation space, removing most constraints on the types of representations we can use. Our approach is an extension of the Representational Similarity Analysis (RSA) method, initially introduced by [Kriegeskorte et al. \(2008\)](#) in the context of understanding neural activation patterns in human brains.

In this work we propose to apply RSA to neural representations of strings from a language on one side, and to structured symbolic representations of these strings on the other side. To capture the similarities between these symbolic representations, we use a *tree kernel*, a metric to compute the proportion of common substructures between trees. This approach enables straightforward comparison of neural and symbolic-linguistic representations. Furthermore, we introduce $\text{RSA}_{\text{REGRESS}}$, a similarity-based analytic method which combines features of RSA and of diagnostic models.

We validate both techniques on neural models which process a synthetic language for arithmetic expressions with a simple syntax and semantics and show that they behave as expected in this controlled setting. We further apply our techniques to two neural models trained on English text, In-fersent ([Conneau et al., 2017](#)) and BERT ([Devlin et al., 2018](#)), and show that both models encode a substantial amount of syntactic information compared to random models and simple bag-of-words representations; we also show that according to our metrics syntax is most salient in the intermediate layers of BERT.

2 Related work

2.1 Analytic methods

The dominance of deep learning models in NLP has brought an increasing interest in techniques to analyze these models and gain insight into how they encode linguistic information. For an overview of analysis techniques, see [Belinkov and Glass \(2019\)](#). The most widespread family of techniques are *diagnostic models*, which use the internal activations of neural networks trained on a particular task as input to another predictive model. The success of such a predictive model is then interpreted as evidence that the predicted information has been encoded by the original neural model. The approach has also been called *auxiliary task* ([Adi et al., 2017](#)), *decoding* ([Alishahi et al., 2017](#)), *diagnostic classifier* ([Hupkes et al., 2018](#)) or *probing* ([Conneau et al., 2018](#)).

Diagnostic models have used a range of predictive tasks, but since their main purpose is to help us better understand the dynamics of a complex model, they themselves need to be kept simple and interpretable. This means that the predicted information in these techniques is typically limited to simple class labels or values, as opposed to symbolic, structured representations of interest to linguists such as syntactic trees. In order to work around this limitation [Tenney et al. \(2019\)](#) present a method for probing complex structures via a formulation named *edge probing*, where classifiers are trained to predict various lexical, syntactic and semantic relations between representation of word spans within a sentence.

Another important consideration when analyzing neural encodings is the fact that a randomly initialized network will often show non-random activation patterns. The reason for this depends on each particular case, but may involve the dynamics of the network itself as well as features of the input data. For a discussion of this issue in the context of diagnostic models see [Zhang and Bowman \(2018\)](#).

Alternative approaches have been proposed to analyzing neural models of language. For example, [Saphra and Lopez \(2019\)](#) train a language model and parallel recurrent models for POS, semantic and topic tagging, and measure the correlation between the neural representations of the language model and the taggers.

Others modify the neural architecture itself to make it more interpretable: [Croce et al.](#)

(2018) adapt layerwise relevance propagation ([Bach et al., 2015](#)) to Kernel-based Deep Architectures ([Croce et al., 2017](#)) in order to retrieve examples which motivate model decisions. A vector representation for a given structured symbolic input is built based on kernel evaluations between the input and a subset of training examples known as landmarks, and the network decision is then traced back to the landmarks which had most influence on it. In our work we also use kernels between symbolic structures, but rather than building a particular interpretable model we propose a general analytical framework.

2.2 Representation Similarity Analysis

[Kriegeskorte et al. \(2008\)](#) present RSA as a variant of pattern-information analysis, to be applied for understanding neural activation patterns in human brains, for example syntactic computations ([Tyler et al., 2013](#)) or sensory cortical processing ([Yamins and DiCarlo, 2016](#)). The core idea is to find connections between data from neuroimaging, behavioral experiments and computational modeling by correlating representations of stimuli in each of these representation spaces via their pairwise (dis)similarities. RSA has also been used for measuring similarities between neural-network representation spaces (e.g. [Bouchacourt and Baroni, 2018](#); [Chrupała, 2019](#)).

2.3 Tree kernels

For extending RSA to a structured representation space, we need a metric for measuring (dis)similarity between two structured representations. Kernels provide a suitable framework for this purpose: [Collins and Duffy \(2002\)](#) introduce convolutional kernels for syntactic parse trees as a metric which quantifies similarity between trees as the number of overlapping tree fragments between them, and introduce a polynomial time algorithm to compute these kernels; [Moschitti \(2006\)](#) propose an efficient algorithm for computing tree kernels in linear average running time.

2.4 Synthetic languages

When developing techniques for analyzing neural network models of language, several studies have used synthetic data from artificial languages. Using synthetic language has the advantage that its structure is well-understood and the complexity of the language and the statistical characteristics of the generated data can be carefully con-

trolled. The tradition goes back to the first generation of connectionist models of language (Elman, 1990; Hochreiter and Schmidhuber, 1997). More recently, Sennhauser and Berwick (2018) and Skachkova et al. (2018) both use context-free grammars to generate data, and train RNN-based models to identify matching numbers of opening and closing brackets (so called Dyck languages). The task can be learned, but Sennhauser and Berwick (2018) report that the models fail to generalize to longer sentences. Paperno (2018) also show that with extensive training and the appropriate curriculum, LSTMs trained on synthetic language can learn compositional interpretation rules.

Nested arithmetic languages are also appealing choices since they have an unambiguous hierarchical structure and a clear compositional semantic interpretation (i.e. the value of the arithmetic expression). Hupkes et al. (2018) train RNNs to calculate the value of such expressions and show that they perform and generalize well to unseen strings. They apply diagnostic classifiers to analyze the strategy employed by the RNN model.

3 Similarity-based analytical methods

RSA finds connections between data from two different representation spaces. Specifically, for each representation type we compute a matrix of similarities between pairs of stimuli. Pairs of these matrices are then subject to second-order analysis by extracting their upper triangulars and computing a correlation coefficient between them.

Thus for a set of objects X , given a similarity function s_k for a representation k , the function S_k which computes the representational similarity matrix is defined as:

$$\begin{aligned} S_k(X) &= \mathbf{U} \\ U_{i,j} &= s_k(X_i, X_j), \end{aligned} \quad (1)$$

and the RSA score between representations k and l for data X is the correlation (such as Pearson’s correlation coefficient r) between the upper triangulars $S_k(X)$ and $S_l(X)$, excluding the diagonals.

Structured RSA We apply RSA to neural representations of strings from a language on one side, and to structured symbolic representations of these strings on the other side. The structural properties are captured by defining appropriate similarity

functions for these symbolic representations; we use tree kernels for this purpose.

A tree kernel measures the similarity between a pair of tree structures by computing the number of tree fragments they share. Collins and Duffy (2002) introduce an algorithm for efficiently computing this quantity; a tree fragment in their formulation is a set of connected nodes subject to the constraint that only complete production rules are included.

Following Collins and Duffy (2002), we calculate the tree kernel between two trees T_1 and T_2 as:

$$K(T_1, T_2) = \sum_{n_1 \in T_1} \sum_{n_2 \in T_2} C(n_1, n_2, \lambda), \quad (2)$$

where n_1 and n_2 are the complete sets of tree fragments in T_1 and T_2 , respectively, and the function $C(n_1, n_2, \lambda)$ is calculated as shown in figure 2. The parameter λ is used to scale the relative importance of tree fragments with their size. Lower values of this parameter discount larger tree fragments in the computation of the kernel; the value 1 does not do any discounting. See Figure 1 for the illustration of the effect of the value of λ on the kernel.

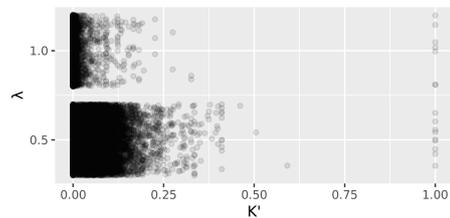


Figure 1: Distribution of values of the tree kernel for two settings of discounting parameter λ , for syntax trees of a sample of English sentences.

We work with normalized kernels: given a function K which computes the raw count of tree fragments in common between trees t_1 and t_2 , the normalized tree kernel is defined as:

$$K'(t_1, t_2) = \frac{K(t_1, t_2)}{\sqrt{K(t_1, t_1)K(t_2, t_2)}}. \quad (3)$$

Figure 3 shows the complete set of tree fragments which the tree kernel implicitly computes for an example syntax tree.

$$C(n_1, n_2, \lambda) = \begin{cases} 0, & \text{if } \text{prod}(n_1) \neq \text{prod}(n_2) \\ \lambda, & \text{if } \text{preterm}(n_1) \wedge \text{preterm}(n_2) \\ \lambda \prod_{i=1}^{\text{nc}(n_1)} (1 + C(\text{ch}(n_1, i), \text{ch}(n_2, i), \lambda)) & \text{otherwise.} \end{cases}$$

Figure 2: Dynamic programming formula for computing a convolution kernel, after Collins and Duffy (2002). Here $\text{nc}(n)$ is the number of children of a given (sub)tree, and $\text{ch}(n, i)$ is its i^{th} child; $\text{prod}(n)$ is the production of node n , and $\text{preterm}(n)$ is true if n is a preterminal node.

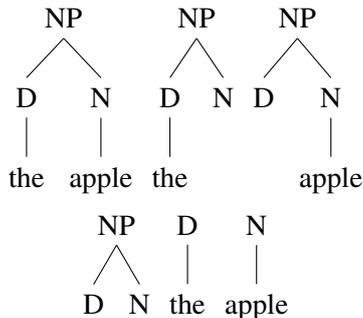


Figure 3: The complete set of tree fragments as defined by the tree kernel for the syntax tree corresponding to *the apple*, after Collins and Duffy (2002).

RSA_{REGRESS} Basic RSA measures correlation between similarities in two different representations globally, i.e. how close they are in their totality. In contrast, diagnostic models answer a more specific question: to what extent a particular type of information can be extracted from a given representation. For example, while for a particular neural encoding of sentences it may be possible to predict the length of the sentence with high accuracy, the RSA between this representation and the strings represented only by their length may be relatively small in magnitude, since the neural representation may be encoding many other aspects of the input in addition to its length.

We introduce **RSA_{REGRESS}**, a method which shares features of both classic RSA as well as the diagnostic model approach. Like RSA it is based on two similarity functions s_k and s_l specific to two different representations k and l . But rather than computing the square matrices $S_k(X)$ and $S_l(X)$ for a set of objects X , we sample a reference set of objects R to act as anchor points, and then embed the objects of interest X in the representation space k via the representational similar-

ity function σ_k defined as:¹

$$\begin{aligned} \sigma_k(X, R) &= \mathbf{V} \\ V_{i,j} &= s_k(X_i, R_j), \end{aligned} \quad (4)$$

Likewise for representation l , we calculate σ_l for the same set of objects X . The rows of the two resulting matrices contain two different views of the objects of interest, where the dimensions of each view indicate the degree of similarity for a particular reference anchor point. We can now fit a multivariate linear regression model to map between the two views:

$$\widehat{\mathbf{B}}, \mathbf{a} = \arg \min_{\mathbf{B}, \mathbf{a}} \text{MSE}(\mathbf{B}\sigma_k(X, R) + \mathbf{a}, \sigma_l(X, R)) \quad (5)$$

where k is the source and l is the target view, and MSE is the mean squared error. The success of this model can be seen as an indication of how predictable representation l is from representation k . Specifically, we use a cross-validated Pearson’s correlation between predicted and true targets for an L_2 -penalized model.

4 Synthetic language

Evaluation of analysis methods for neural network models is an open problem. One frequently resorts to largely qualitative evaluation: checking whether the conclusions reached via a particular approach have face validity and match pre-existing intuitions. However pre-existing intuitions are often not reliable when it comes to complex neural models applied to also very complex natural language data. It is helpful to simplify one part of the overall system and apply the analytic technique of interest on a neural model which processes a simple and well-understood synthetic language. As our first case study, we use a simple language of

¹Note that σ_k is simply a generalization of S_k to the non-square case, namely $S_k(X) = \sigma_k(X, X)$.

Syntax	Meaning
$E \rightarrow L E_1 O E_2 R$	$[E] = [O]([E_1], [E_2])$
$E \rightarrow D$	$[E] = [D]$
$O \rightarrow +$	$[O] = \lambda x, y. x + y \bmod 10$
$O \rightarrow -$	$[O] = \lambda x, y. x - y \bmod 10$
$L \rightarrow ($	
$R \rightarrow)$	
$D \rightarrow 0$	$[D] = 0$
\vdots	\vdots
$D \rightarrow 9$	$[D] = 9$

Table 1: Grammar $G(L)$ of a language L expressing addition and subtraction modulo 10 in infix notation. The notation $[\cdot]$ stands for the semantic evaluation function. Subscripts on symbols serve to distinguish their multiple occurrence.

arithmetic expressions. Here we first describe the language and its syntax and semantics, and then introduce neural recurrent models which process these expressions.

4.1 Arithmetic expressions

Our language consists of expressions which encode addition and subtraction modulo 10. Consider the example expression $((6+2) - (3+7))$. In order to evaluate the whole expression, each parenthesized sub-expression is evaluated modulo 10: in this case the left sub-expression evaluates to 8, the right one to 0 and the whole expression to 8. Table 1 gives the context-free grammar which generates this language, and the rules for semantic evaluation. Figure 4 shows the syntax tree for the example expression according to this grammar. This language lacks ambiguity, has a small vocabulary (14 symbols) and simple semantics, while at the same time requiring the processing of hierarchical structure to evaluate its expressions.²

Generating expressions In order to generate expressions in L we use the recursive function GENERATE defined in Algorithm 1. The function receives two input parameters: the branching probability p and the decay factor d . In the recursive call to GENERATE in lines 4 and 5 the probability p is divided by the decay factor. Larger values of d lead to the generation of smaller expressions. Within the branching path in line 6 the operator is selected uniformly at random, and likewise in the non-branching path in line 9 the digit is sampled uniformly.

²The grammar is more complex than strictly needed in order to facilitate the computation of the Tree Kernel, which assumes each vocabulary symbol is expanded from a pre-

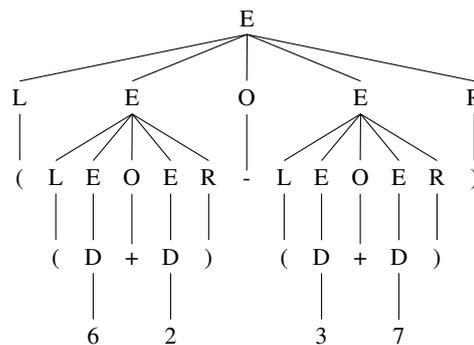


Figure 4: Syntax tree of the expression $((6+2) - (3+7))$.

Algorithm 1 Recursive function for generating an expression of language L .

```

1: function GENERATE( $p, d$ )
2:    $\text{branch} \sim \text{BERNOULLI}(p)$ 
3:   if  $\text{branch}$  then
4:      $e_1 \leftarrow \text{GENERATE}(p/d, d)$ 
5:      $e_2 \leftarrow \text{GENERATE}(p/d, d)$ 
6:      $\text{op} \sim \text{UNIFORM}([+, -])$ 
7:     return  $[E [L ( ] e_1 [O \text{op} ] e_2 [R ) ]]$ 
8:   else
9:      $\text{digit} \sim \text{UNIFORM}([0, \dots, 9])$ 
10:    return  $[E [D \text{digit} ]]$ 
11:  end if
12: end function

```

4.2 Neural models of arithmetic expressions

We define three recurrent models which process the arithmetic expressions from language L . Each of them is trained to predict a different target, related either to the syntax of the language or to its semantics. We use these models as a testbed for validating our analytical approaches. All these models share the same recurrent encoder architecture, based on LSTM (Hochreiter and Schmidhuber, 1997).

Encoder The encoder consists of a trainable embedding lookup table for the input symbols, and a single-layer LSTM. The state of the hidden layer of the LSTM at the last step in the sequence is used as a representation of the input expression.

SEMANTIC EVALUATION This model consists of the encoder as described above, which passes its representation of the input to a multi-layer perceptron component with a single output neuron. It is trained to predict the value of the input expression, with mean squared error as the loss function. In order to perform this task we would expect that the model needs to encode the hierarchical structural node.

ture of the expression to some extent while also encoding the result of actually carrying out the operations of semantic evaluation.

TREE DEPTH This model is similar to SEMANTIC EVALUATION but is trained to predict the depth of the syntax tree corresponding to the expression instead of its value. We expect this model to need to encode a fair amount of hierarchical information, but it can completely ignore the semantics of the language, including the identity of the digit symbols.

INFIX-TO-PREFIX This model uses the encoder to create a representation of the input expression, which it then decodes in its prefix form. For example, the expression $((6+2) - (3+7))$ is converted to $(- (+62) (+37))$. The decoder is an LSTM trained as a conditional language model, i.e. its initial hidden state is the output of the encoder and its input at each step is the embedding of previous output symbol. The loss function is categorical cross-entropy. We would expect this model to encode the hierarchical structure in some form as well as the identity of the digit symbols, but it can ignore the compositional semantics of the language.

4.3 Reference representations

We use RSA to correlate the neural encoders from Section 4.2 with reference syntactic and semantic information about the arithmetic expressions. For the neural representations we use cosine distance as the dissimilarity metric. The reference representations and their associated dissimilarity metrics are described below.

Semantic value This is simply the value to which each expression evaluates, also used as the target of the SEMANTIC EVALUATION model. As a measure of dissimilarity we use the absolute difference between values, which ranges from 0 to 9.

Tree depth This is the depth of the syntax tree for each expression, also used as the target of the TREE DEPTH model. We use the absolute difference as the dissimilarity measure. The dissimilarity is minimum 0 and has no upper bound, but in our data the typical maximum value is around 7.

Tree kernel This is an estimate of similarity between two syntax trees based on the number of tree fragments they share, as described in Section 3. The normalized tree kernel metric ranges between

0 and 1, which we convert to dissimilarity by subtracting it from 1.

The semantic value and tree depth correlates are easy to investigate with a variety of analytic methods including diagnostic models; we include them in our experiments as a point of comparison. We use the tree kernel representation to evaluate structured RSA for a simple synthetic language.

4.4 Experimental settings

We implement the neural models in PyTorch 1.0.0. We use the following model architecture: encoder embedding layer size 64, encoder LSTM size 128, for the regression models, MLP with 1 hidden layer of size 256; for the sequence-to-sequence model the decoder hyper-parameters are the same as the encoder. The symbols are predicted via a linear projection layer from hidden state, followed by a softmax. Training proceeds following a curriculum: we first train on 100,000 batches of size 32 of random expressions sampled with decay $d = 2.0$, followed by 200,000 batches with $d = 1.8$ and finally 400,000 batches with $d = 1.5$. We optimize with Adam with learning rate 0.001. We report results on expressions sampled with $d = 1.5$. See Figure 5 for the distribution of expression sizes for these values of d .

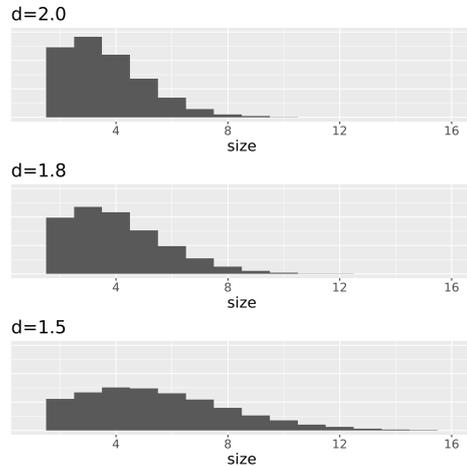


Figure 5: Distribution of expression sizes when varying the value of the decay parameter d . The size of an expression is measured as the number of its digit nodes.

We report all results for two conditions: randomly initialized, and trained, in order to quantify the effect of learning on the activation patterns. The trained model is chosen by saving model weights during training every 10,000 batches and selecting the weights with the smallest loss on 1,000 held-out validation expressions. Results

are reported on separate test data consisting of 2,000 expressions and 200 reference expressions for $\text{RSA}_{\text{REGRESS}}$ embedding.

4.5 Results

Table 2 shows the results of our experiments, where each row shows a different encoder type and each column a different target task.

Semantic value and tree depth As a first sanity check, we would like to see whether the RSA techniques show the same pattern captured by the diagnostic models. As expected, both diagnostic and RSA scores are the highest when the objective function used to train the encoder and the analytical reference representations match: for example, the SEMANTIC EVALUATION encoder scores high on the *semantic value* reference, both for the diagnostic model and the RSA. Furthermore, the scores for the value and depth reference representation according to the diagnostic model and according to $\text{RSA}_{\text{REGRESS}}$ are in agreement. The scores according to RSA in some cases show a different picture. This is expected, as RSA answers a substantially different question than the other two approaches: it looks at how the whole representations match in their similarity structure, whereas both the diagnostic model and $\text{RSA}_{\text{REGRESS}}$ focus on the part of the representation that encodes the target information the strongest.

Tree Kernel We can use both RSA and $\text{RSA}_{\text{REGRESS}}$ for exploring whether the hidden activations encode any structural representation of syntax: this is evident in the scores yielded by the TK reference representations. As expected, the highest scores for both methods are gained when using INFIX-TO-PREFIX encodings, the task that relies the most on the hierarchical structure of an input string. $\text{RSA}_{\text{REGRESS}}$ yields the second-highest score for TREE DEPTH encodings, which also depend on aspects of tree structure. The overall pattern for the TK with different values of the discounting parameter λ is similar, even though the absolute values of the scores vary. What is unexpected is the results for the random encoder, which we turn to next.

Random encoders The non-random nature of the activation patterns of randomly initialized models (e.g., Zhang and Bowman, 2018) is also strongly in evidence in our results. For example the random encoder has quite a high score for

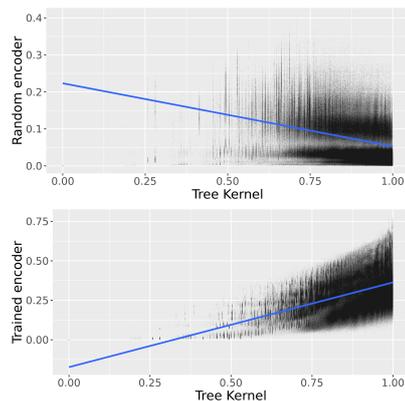


Figure 6: Scatterplot of dissimilarity values according to random encoder or trained INFIX-TO-PREFIX encoder and the Tree Kernel ($\lambda = 0.5$)

diagnostic regression on *tree depth*. Even more striking is the fact that the random encoder has substantial *negative* RSA score for the Tree Kernel: thus, expression pairs more similar according to the Tree Kernel are less similar according to the random encoder, and vice-versa.

When applying RSA we can inspect the full correlation pattern via a scatter-plot of the dissimilarities in the reference and encoder representations. Figure 6 shows the data for the random encoder and the Tree Kernel representations. As can be seen, the negative correlation for the random encoder is due to the fact that according to the Tree Kernel, expression pairs tend to have high dissimilarities, while according to the random encoder’s activations they tend to have overall low dissimilarities. For the trained INFIX-TO-PREFIX encoder the dissimilarities are clearly positively correlated with the TK dissimilarities.

Thus the raw correlation value for the trained encoder is a biased estimate of the effect of learning, as learning has to overcome the initially substantial negative correlation: a better estimate is the difference between scores for the learned and random model. It is worth noting that the same approach would be less informative for the diagnostic model approach or for $\text{RSA}_{\text{REGRESS}}$. For a regression model the correlation scores will be positive, and when taking the difference between learned and random scores, they may cancel out, even though a particular information may be predictable from the random activations in a completely different way than from the learned activations. This is what we see for the $\text{RSA}_{\text{REGRESS}}$ scores for random vs. INFIX-TO-PREFIX encoder:

Encoder	Loss	Diagnostic		RSA				RSA _{REGRESS}			
		Value	Depth	Value	Depth	TK(1)	TK(0.5)	Value	Depth	TK(1)	TK(0.5)
RANDOM		0.01	0.80	0.01	0.23	-0.24	-0.33	-0.01	0.57	0.41	0.63
SEMANTIC EVAL.	0.07	0.97	0.70	0.62	0.05	0.02	0.01	0.97	0.55	0.38	0.61
TREE DEPTH	0.00	-0.03	1.00	0.01	0.72	0.10	-0.06	-0.03	0.97	0.49	0.87
INFIX-TO-PREFIX	0.00	0.02	0.97	-0.00	0.64	0.35	0.53	0.02	0.88	0.58	0.96

Table 2: Scores for diagnostic regression, RSA, and RSA_{REGRESS} with respect to expression value, expression tree depth and the Tree Kernel (TK) with $\lambda = 1$ and $\lambda = 0.5$. All scores are Pearson’s correlation coefficients. For the diagnostic model and RSA_{REGRESS} they are cross-validated correlations between target and predicted values. The randomly initialized encoder is the same for all encoder types, and thus there is only a single row for the RANDOM encoder. The loss column shows the loss of the full model on the test data: mean squared error for SEMANTIC EVALUATION and TREE DEPTH, and cross-entropy for INFIX-TO-PREFIX.

the scores partially cancel out, and given the pattern in Figure 6 it is clear that subtracting them is misleading. It is thus a good idea to complement the RSA_{REGRESS} score with the plain RSA correlation score in order to obtain a full picture of how learning affects the neural representations.

Overall, these results show that RSA_{REGRESS} can be used to answer the same sort of questions as the diagnostic model. It has the added advantage of being also easily applicable to structured symbolic representations, while the RSA scores and the full RSA correlation pattern provides a complementary source of insight into neural representations. Encouraged by these findings, we next apply both RSA and RSA_{REGRESS} to representations of natural language sentences.

5 Natural language

Here we use our proposed RSA-based techniques to compare tree-structure representations of natural language sentences with their neural representations captured by sentence embeddings. Such embeddings are often provided by NLP systems trained on unlabeled text, using variants of a language modeling objective (e.g. Peters et al., 2018), next and previous sentence prediction (Kiros et al., 2015; Logeswaran and Lee, 2018), or discourse based objectives (Nie et al., 2017; Jernite et al., 2017). Alternatively they can be either fully trained or fine-tuned on annotated data using a task such as natural language inference (Conneau et al., 2017). In our experiments we use one of each type of encoders.

5.1 Encoders

Bag of words As a baseline we use a classic bag of words model where a sentence is represented by a vector of word counts. We do not exclude

any words and use raw, unweighted word counts.

InferSent This is the supervised model described in Conneau et al. (2017) based on a bidirectional LSTM trained on natural language inference. We use the `infernent2` model with pre-trained fastText (Bojanowski et al., 2017) word embeddings.³ We also test a randomly initialized version of this model, including random word embeddings.

BERT This is an unsupervised model based on the Transformer architecture (Vaswani et al., 2017) trained on a cloze-task and next-sentence prediction (Devlin et al., 2018). We use the Pytorch version of the large 24-layer model (`bert-large-uncased`).⁴ We also test a randomly initialized version of this model.

5.2 Experimental settings

Data We use a sample of data from the English Web Treebank (EWT) (Bies et al., 2012) which contains a mix of English weblogs, newsgroups, email, reviews and question-answers manually annotated for syntactic constituency structure. We use the 2,002 sentences corresponding to the development section of the EWT Universal Dependencies (Silveira et al., 2014), plus 200 sentences from the training section as reference sentences when fitting RSA_{REGRESS}.

Tree Kernel Prior to computing the Tree Kernel scores we delexicalize the constituency trees by replacing all terminals (i.e. words) with a single placeholder value X . This ensures that only syntactic structure, and not lexical overlap, contributes to kernel scores. We compute kernels for the values of $\lambda \in \{1, \frac{1}{2}\}$.

³Available at <https://github.com/facebookresearch/InferSent>.

⁴Available at <https://github.com/huggingface/pytorch-pretrained-BERT>.

Encoder	Train	λ	RSA	$\text{RSA}_{\text{REGRESS}}$
BoW		0.5	0.18	0.50
InferSent	−	0.5	0.24	0.51
BERT last	−	0.5	0.12	0.49
BERT best	−	0.5	0.14	0.53
InferSent	+	0.5	0.30	0.71
BERT last	+	0.5	0.16	0.59
BERT best	+	0.5	0.32	0.70
<hr/>				
BoW		1.0	-0.01	0.40
InferSent	−	1.0	0.00	0.48
BERT last	−	1.0	-0.08	0.50
BERT best	−	1.0	-0.07	0.52
InferSent	+	1.0	0.10	0.59
BERT last	+	1.0	0.03	0.53
BERT best	+	1.0	0.18	0.60

Table 3: Correlation scores for encoders against Tree Kernel with varying λ . Scores for both RSA and $\text{RSA}_{\text{REGRESS}}$ are Pearson’s r . The column *Train* indicates whether the encoder (including the word embeddings) is randomly initialized (−), or trained (+). For BERT, we report scores for the topmost (last) layer and for the layer which maximizes the given score (best).

Embeddings For the BERT embeddings we use the vector associated with the first token (CLS) for a given layer. For InferSent, we use the default max-pooled representation.

Fitting When fitting $\text{RSA}_{\text{REGRESS}}$ we use L2-penalized multivariate linear regression. We report the results for the value of the penalty $= 10^n$, for $n \in \{-3, -2, -1, 0, 1, 2\}$, with the highest 10-fold cross-validated Pearson’s r between target and predicted similarity-embedded vectors.

5.3 Results

Table 3 shows the results of applying RSA and $\text{RSA}_{\text{REGRESS}}$ on five different sentence encoders, using the Tree Kernel reference. Results are reported using two different values for the Tree Kernel parameter λ .

As can be seen, with $\lambda = \frac{1}{2}$, all the encoders show a substantial RSA correlation with the parse trees. The highest scores are achieved by the trained InferSent and BERT, but even Bag of Words and untrained versions of InferSent and BERT show a sizeable correlation with syntactic trees according to both RSA and $\text{RSA}_{\text{REGRESS}}$.

When structure matching is strict ($\lambda = 1$), only trained BERT and InferSent capture syntactic information according to RSA; however, $\text{RSA}_{\text{REGRESS}}$ still shows moderate correlation for BoW and the untrained versions of BERT and InferSent. Thus $\text{RSA}_{\text{REGRESS}}$ is less sensitive to the value of λ than RSA since changing it from $\frac{1}{2}$ to 1

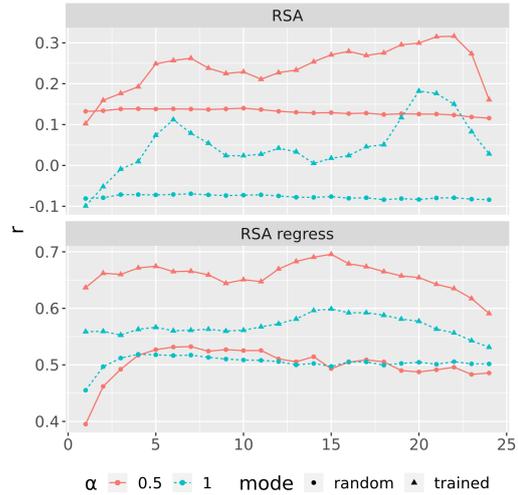


Figure 7: RSA and $\text{RSA}_{\text{REGRESS}}$ scores for embeddings from all the layers of BERT vs Tree Kernel for two values of λ . Both randomly initialized and trained versions of BERT are shown. The embeddings are vectors at the first token (CLS) at each layer.

does not alter results in a qualitative sense.

Figure 7 shows how RSA and $\text{RSA}_{\text{REGRESS}}$ scores change when correlating Tree Kernel estimates with embeddings from different layers of BERT. For trained models, scores peak between layers 15–22 (depending on metric and λ) and decline thereafter, which indicates that the final layers are increasingly dedicated to encoding aspects of sentences other than pure syntax.

6 Conclusion

We present two RSA-based methods for correlating neural and syntactic representations of language, using tree kernels as a measure of similarity between syntactic trees. Our results on arithmetic expressions confirm that both versions of structured RSA capture correlations between different representation spaces, while providing complementary insights. We apply the same techniques to English sentence embeddings, and show where and to what extent each representation encodes syntactic information. The proposed methods are general and applicable not just to constituency trees, but given a similarity metric, to any symbolic representation of linguistic structures including dependency trees or Abstract Meaning Representations. We plan to explore these options in future work. A toolkit with the implementation of our methods is available at <https://github.com/gchrupala/ursa>.

References

- Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. *International Conference on Learning Representations (ICLR)*.
- Afra Alishahi, Marie Barking, and Grzegorz Chrupała. 2017. Encoding of phonology in a recurrent neural model of grounded speech. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 368–378.
- Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS one*, 10(7):e0130140.
- Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.
- Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. 2012. English Web Treebank LDC2012T13. Web Download.
- Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.
- Diane Bouchacourt and Marco Baroni. 2018. [How agents see things: On visual representations in an emergent language game](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 981–985, Brussels, Belgium. Association for Computational Linguistics.
- Grzegorz Chrupała. 2019. Symbolic inductive bias for visually grounded learning of spoken language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.
- Michael Collins and Nigel Duffy. 2002. Convolution kernels for natural language. In *Advances in neural information processing systems*, pages 625–632.
- Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. [Supervised learning of universal sentence representations from natural language inference data](#). In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.
- Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. [What you can cram into a single vector: Probing sentence embeddings for linguistic properties](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136. Association for Computational Linguistics.
- Daniilo Croce, Simone Filice, Giuseppe Castellucci, and Roberto Basili. 2017. Deep learning in semantic kernel spaces. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 345–354.
- Daniilo Croce, Daniele Rossini, and Roberto Basili. 2018. [Explaining non-linear classifier decisions within kernel-based deep architectures](#). In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 16–24. Association for Computational Linguistics.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. [BERT: pre-training of deep bidirectional transformers for language understanding](#). *CoRR*, abs/1810.04805.
- Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. 2018. Visualisation and ‘diagnostic classifiers’ reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926.
- Yacine Jernite, Samuel R Bowman, and David Sonntag. 2017. Discourse-based objectives for fast unsupervised sentence representation learning. *arXiv preprint arXiv:1705.00557*.
- Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.
- Nikolaus Kriegeskorte, Marieke Mur, and Peter A Bandettini. 2008. Representational similarity analysis—connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4.
- Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. In *International Conference on Learning Representations*.
- Alessandro Moschitti. 2006. Making tree kernels practical for natural language learning. In *11th conference of the European Chapter of the Association for Computational Linguistics*.

- Allen Nie, Erin D Bennett, and Noah D Goodman. 2017. Dissent: Sentence representation learning from explicit discourse relations. *arXiv preprint arXiv:1710.04334*.
- Denis Paperno. 2018. [Limitations in learning an interpreted language with recurrent models](#). In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 384–386. Association for Computational Linguistics.
- Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 2227–2237.
- Naomi Saphra and Adam Lopez. 2019. Understanding learning dynamics of language models with SVCCA. In *Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*. Association for Computational Linguistics.
- Luzi Sennhauser and Robert Berwick. 2018. [Evaluating the ability of lstms to learn context-free grammars](#). In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 115–124. Association for Computational Linguistics.
- Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.
- Natalia Skachkova, Thomas Trost, and Dietrich Klakow. 2018. [Closing brackets with recurrent neural networks](#). In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 232–239. Association for Computational Linguistics.
- Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, et al. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. In *ICLR 2019*.
- Lorraine Komisarjevsky Tyler, Teresa PL Cheung, Barry J Devereux, and Alex Clarke. 2013. Syntactic computations in the language network: characterizing dynamic network properties using representational similarity analysis. *Frontiers in psychology*, 4:271.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.
- Daniel LK Yamins and James J DiCarlo. 2016. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356.
- Kelly Zhang and Samuel Bowman. 2018. Language modeling teaches you more syntax than translation does: Lessons learned through auxiliary task analysis. *arXiv preprint arXiv:1809.10040*.