Diagnostic Classifiers: Revealing how Neural Networks Process Hierarchical Structure

Veldhoen, S.; Hupkes, D.; Zuidema, W.

Publication date
2016
Document Version
Final published version
Published in
CoCo 2016 : Cognitive Computation

Diagnostic classifiers: revealing how neural networks process hierarchical structure

Sara Veldhoen, Dieuwke Hupkes and Willem Zuidema
ILLC, University of Amsterdam
P.O.Box 94242, 1090 CE Amsterdam, Netherlands
{s.f.veldhoen, d.hupkes, zuidema}@uva.nl

Abstract

We investigate how neural networks can be used for hierarchical, compositional semantics. To this end, we define the simple but nontrivial artificial task of processing nested arithmetic expressions and study whether different types of neural networks can learn to add and subtract. We find that recursive neural networks can implement a generalising solution, and we visualise the intermediate steps: projection, summation and squashing. We also show that gated recurrent neural networks, which process the expressions incrementally, perform surprisingly well on this task: they learn to predict the outcome of the arithmetic expressions with reasonable accuracy, although performance deteriorates with increasing length. To analyse what strategy the recurrent network applies, visualisation techniques are less insightful. Therefore, we develop an approach where we formulate and test hypotheses on what strategies these networks might be following. For each hypothesis, we derive predictions about features of the hidden state representations at each time step, and train 'diagnostic classifiers' to test those predictions. Our results indicate the networks follow a strategy similar to our hypothesised 'incremental strategy'.

1 Introduction

A key property of language is its hierarchical compositional semantics: the meaning of larger wholes such as phrases and sentences depends on the meaning of the words they are composed of and the way they are put together, and phrases can be hierarchically combined into more complex phrases. For example, the meaning of the compound noun *natural language research paper* is a combination of the meanings of *natural language* and *research paper*, which in turn are combinations of the meanings of the individual words. Whether 'classical' neural networks can adequately compute the grammaticality and meaning of sentences with hierarchical structure has been a the topic of many heated debates in linguistics and cognitive science [e.g. Pinker and Mehler, 1990, Fodor and Pylyshyn, 1988].

In the literature, one can find many attempts at designing neural models that are able to capture hierarchical compositionality [e.g. Elman, 1990, Rodriguez, 2001, Martens, 2011], as well as a vast amount of papers on improved computational methods to train such models [e.g. Zeiler, 2012, Kingma and Ba, 2014], but very few focus on understanding the internal dynamics of the parameter-rich models [but see Karpathy et al., 2015]. In this paper, we do not focus on designing better training algorithms or models, or even on achieving high accuracies, but rather on understanding *how* neural networks can implement solutions that involve hierarchical compositionality.

A substantial problem encountered in such an undertaking, is the difficulty of evaluating whether a neural model can in fact capture the semantics of a sentence, as meanings can usually not be summarised by a single number or class. Several tools have been developed to assess the quality of

```
result_stack = [], mode_stack = [];
result = 0, mode = +;
for symbol in expression do
    if symbol == '(' then
        mode_stack.append(mode);
        result_stack.append(result);
        result = 0
    else if symbol == ')' then
        mode = mode_stack.pop();
        prev_result = result_stack.pop();
        result = mode(prev_result, result)
    else if symbol == '+' then
        mode = +
    else if symbol == '-' then
        mode = -
    else
        result = apply(mode, result, symbol)
end
return result
```

(a) Recursive strategy

```
mode_stack = [];
result = 0, mode = +;
for symbol in expression do
    if symbol == '(' then
        mode_stack.append(mode)
    else if symbol == ')' then
        mode = mode_stack.pop()
    else if symbol == '+' then
        pass
    else if symbol == '-' then
        if mode == - then
            mode = +
        else
            mode = -
    else
        result = apply(mode, result, symbol)
end
return result
```

(b) Incremental strategy

Figure 1: Different symbolic strategies for incrementally solving arithmetic expressions incrementally. The `apply` function applies the operator specified by *mode* $(+, -)$ to the two numbers specified by *result* and *symbol*.

lexical representations, but no satisfactory option is available to study representations that result from composition. A common approach to train and evaluate sentence representations is to have them execute a task that requires semantic knowledge of the sentence such as sentiment analysis, logical inference or question-answering [e.g. Le and Zuidema, 2015a,b, Hermann et al., 2015].

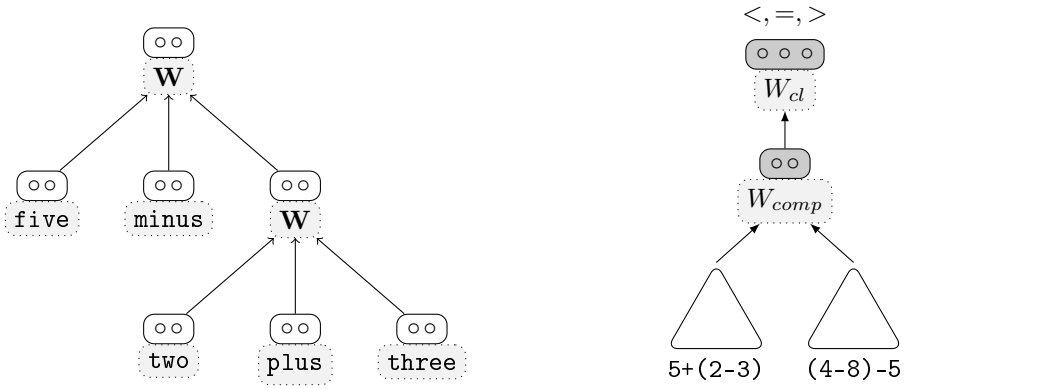
Although solving such tasks might be worth pursuing in its own right, this task-driven setting obscures the mechanisms for semantic composition that we aim to investigate. Therefore, in this paper we take a different approach, by looking at an artificial language of which both the sentences and the lexical items have clearly (and symbolically) defined meanings. We present a thorough analysis of two different kinds of neural models that process sentences from this language, as well as an innovative method to study their behaviour: the diagnostic classifier.

2 Arithmetic language

The language we consider for this study consists of words for all integers in the range $\{-10, \ldots, 10\}$, the operators `+` and `-` and the brackets `(` and `)`. All (bracketed) mathematically correct expressions that can be formed with these words constitute the sentences of the arithmetic language. The (compositional) meaning of a sentence is the solution of the corresponding arithmetic expression. As the minus operator is non-commutative, meanings depend not only on lexical items, but also on the syntactic structure as prescribed by the brackets. The unambiguously defined syntax and semantics, as well as their entanglement, make the arithmetic language very suitable to evaluate whether and how neural network models can learn compositional semantics.

Naturally, we do not aim to find or improve ways to solve the task of computing the outcome of arithmetic expressions. Rather, we study the solutions found by neural networks, using our understanding of how the task can be solved symbolically. In what follows, we will distinguish sentences by length, that is: 'L7' refers to those sentences that contain exactly 7 digits. We also distinguish subsets containing only left-branching (e.g. L9L) or right-branching expressions (L6R), respectively, as well as syntactically non-restricted subsets.

Symbolic strategies One can think of different strategies to incrementally compute the solution of an arithmetic expression, the perhaps most obvious of which involves recursively traversing through the expression and computing the outcome of all subtrees before finally accumulating the results at the end. Alternatively, the digits can be accumulated immediately at the moment they are encountered, maintaining a prediction of the solution at any point during the computation. Figure 1



(a) A TreeRNN to represent `( 5 + ( 2 - 3 ) )`

(b) Training setup: compare two expressions from concatenation of root representations

Figure 2: Network architecture TreeRNN

contains procedural descriptions for both the recursive and the incremental strategy. An incremental implementation of the recursive strategy requires a stack to store previously computed outcomes of subtrees, as well as the operators needed to integrate them. The incremental strategy, on the other hand, requires only information about previously seen operators, which is less demanding.

3 Models

We study two types of network architectures: recursive neural networks and recurrent neural networks. The former are structured following the syntactic structure of a sentence, whereas the latter processes sentences sequentially, reading them one word at the time.

3.1 Recursive Neural Network (TreeRNN)

The Recursive Neural Network or *TreeRNN* [Socher et al., 2010, Goller and Kuechler, 1996] is a hybrid symbolic-connectionist model that explicitly deals with compositionality by shaping its architecture after the syntactic structure of its input. This structure is thus assumed to be known beforehand, when the network is instantiated, which makes the network dependent on external parses. Figure 2a shows a TreeRNN instantiation of an L3 expression.

The composition function of the TreeRNN

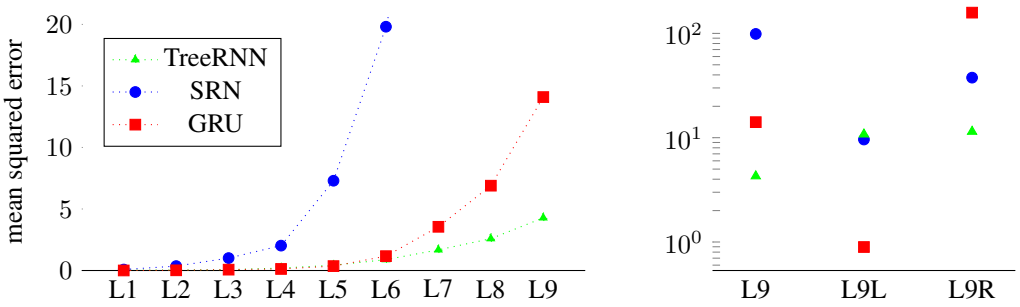
$$\mathbf{p} = \tanh(\mathbf{W} \cdot [\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3] + \mathbf{b}), \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{d \times 3d}$ and $\mathbf{b} \in \mathbb{R}^d$, operates on the concatenation of three d-dimensional vectors $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ that represent subphrases or words. The result is a vectorial parent representation $\mathbf{p}$ of the same dimensionality d. As such, the function can be applied recursively. Notably, there is a single composition function that is applied for each subtree, where the operators are treated as lexical items.

Training The composition function is trained in a regime inspired by Bowman et al. [2015b], who train a network for logical reasoning. Two sentence meanings are computed by treeRNNs with shared parameters. An additional neural classifier predicts the relation between the concatenated sentence representations, which generates an error signal that is backpropagated to update the parameters. In this case, the prediction concerns the (in)equality ('<', '=' or '>') that holds between the two (numerical) solutions. This training setup is displayed in Figure 2b.

3.2 Recurrent Neural Network (RNN)

In addition to the syntactically informed TreeRNNs, we study the behaviour of their well-known sequential counterparts: recurrent neural networks (RNNs). In particular, we consider two types of RNNs: simple recurrent networks (SRNs), as originally introduced by Elman [1990], and Gated



(a) Performance (MSE) with respect to sentence length

(b) MSE (log scale) on L9 and L9L (only left-branching) and L9R (only right-branching)

Figure 3: Mean Squared Error of scalar prediction for best performing models on the test sentences.

Recurrent Units (GRUs), defined in [Cho et al., 2014]. The SRN consist of a single hidden layer $\mathbf{h}$ with a nonlinear activation function and a feedback connection:

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{\mathbf{t-1}} + \mathbf{b}) \tag{2}$$

The GRU is an extended version of this model, in which *gates* are used to modulate the recurrent information flow. Two gate values are computed from the previous hidden layer activation and the current input. The reset gate $\mathbf{r}$ affects the computation of a candidate hidden state $\tilde{\mathbf{h}}_t$:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r) \qquad \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}_h(\mathbf{r} \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{3}$$

where $\odot$ denotes the element wise product and σ the logistic sigmoid function. The new activation $\mathbf{h}_{t-1}$ is a weighted sum of the candidate activation $\tilde{\mathbf{h}}_t$ and the previous activation $\mathbf{h}_{t-1}$, modulated by the update gate $\mathbf{z}$.

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z) \qquad \mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t, \tag{4}$$

All $\mathbf{W} \in \mathbb{R}^{h \times d}$, all $\mathbf{U} \in \mathbb{R}^{h \times h}$ and all $\mathbf{b}$, $\mathbf{z}$ and $\mathbf{r} \in \mathbb{R}^h$ with $h = |\mathbf{h}|$ and $d = |\mathbf{x}|$.

Training The RNNs have access to the syntactic structure implicitly, through the brackets, that are presented to the network as words. Both the embeddings and the weights for the sequential models are trained on an error signal from a simple linear perceptron that predicts the real-valued solution of the expression from the hidden representation in the last time step.

4 Data and Results

We train all models on an arbitrarly sampled set of expressions from L1, L2, L4, L5 and L7 (3000 expressions from each subset) with randomly generated syntactic structures. During training we update also the word embeddings. All models are trained using stochastic gradient descent with minibatches. The scalar prediction models are trained with optimiser Adam [Kingma and Ba, 2014] and `mse` loss, the original TreeRNNs with Adagrad [Zeiler, 2012] and cross-entropy.[1]

The TreeRNN obtains a classification accuracy of 0.97 on test data (sentences up to length 9) for the comparison task. To make its results comparable to those of the sequential RNNs, we train a neural network with one hidden layer to perform the same task as the RNNs, i.e., predicting the solution of expressions. Importantly, we keep both the embeddings and the parameters of the TreeRNN fixed during this training, and only update the weights of the prediction network.

The main purpose of quantifying how well the task is performed by different types networks is to establish which models learned to perform the task well enough to analyse *how* they implement the

[1]Source code of the simulations can be found at `https://github.com/dieuwkehupkes/Arithmetics`. For part of our implementation we used the Python library Keras [Chollet, 2015].

solution. We therefore only report the results of the models that perform best, which can be found in Figure 3.[2] The SRNs do not seem to have learned a principled solution with this training regime. We find that the TreeRNN generalises adequately to longer (unseen) sentences, evident from the smooth progression of the performance with respect to length [the criterion used by Bowman et al., 2015a]. Although the generalisation capacity of the GRU (with $|h| = 15$) is weaker, it seems the GRU has also learned a solution that is sensitive to the syntactic structure. Figure 3b reveals that completely right-branching sentences are much harder to process for the GRU. Left-branching sentences, which allow for direct accumulation of numbers, prove much easier. The performance of TreeRNNs, on the other hand, suffers from tree depth, regardless of the branching direction.

5 Analysis

In the previous section, we established that a number of our trained networks can - at least to some extent - compute the compositional meaning of sentences and phrases in the arithmetic language in a generalising way. We will present an analysis of the internal dynamics of the best performing models, explaining *how* the task of computing the outcome of the arithmetic expressions is implemented. We will start with considering the TreeRNN, whose extremely low dimensionality allows for a comprehensible account of the implementation of the solution. Subsequently, we will analyse the internal dynamics of the GRU network, and introduce a new approach for testing hypotheses about the strategies a network might engage.

5.1 TreeRNN

To understand how a TreeRNN computes the composition of digits and operators, we break down the computation into different steps: project, sum and squash.[3]

Project The application function of the TreeRNN (equation 1), with $\mathbf{W} = [\mathbf{W}_L; \mathbf{W}_M; \mathbf{W}_R]$, can be rewritten as a sum of three terms, which results in the following expression for the composition function:

$$\mathbf{p} = f(\mathbf{W}_L \cdot \mathbf{x}_1 + \mathbf{W}_M \cdot \mathbf{x}_2 + \mathbf{W}_R \cdot \mathbf{x}_3 + \mathbf{b}) \tag{5}$$

where each $\mathbf{W}_i \in \mathbb{R}^{2\times 2}$ projects one of the three input children to a new position in the two dimensional representational space. Figure 4 depicts the digit embeddings and their projections, illustrating how the left and right children are projected to almost orthogonal subspaces.

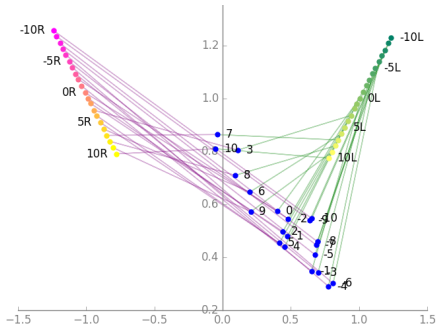
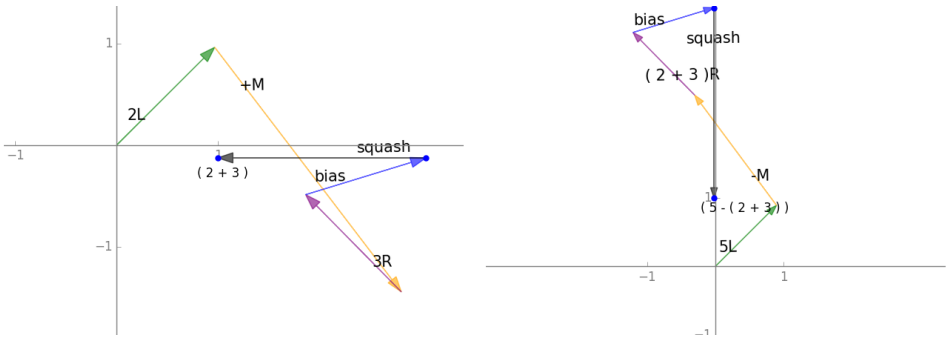


Figure 4: The projections of the seemingly unorganised digit embeddings (blue) through $\mathbf{W}_L$ (as left child, green) and $\mathbf{W}_R$ (as right child, purple) reveal their organisation.

Sum The coloured arrows in Figure 5 show how the projected representations and the bias term are summed. From the two subfigures, it is clear that the projections of the operators + and - through $\mathbf{W}_M$ are roughly opposite. Together with the bias, the result is sent to positions close to the y- and x-axis, for + and - respectively, which is important for the last step of the computation.

[2]Except for the SRN models, the results over different runs were comparable on a performance level.

[3]This analysis was part of the master thesis of Sara Veldhoen [2015].

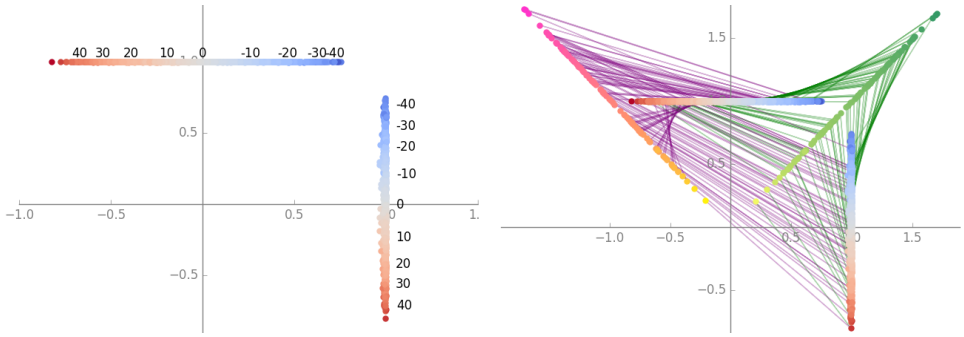


(a) (2 + 3) (b) (5 - ‘(2 + 3)’)

Figure 5: Summation of projections - left (green), middle (yellow) and right (purple) child - together with bias term (blue). Squash ($\tanh$ activation, grey) shifts the result back to the range $(-1, 1)$.

Squash In this last step, both coordinates of the representation are squashed into the range $(-1, 1)$ by the activation function `tanh`. As the projected representations are already within this range for one of the dimensions, depending on the operator, the effect of the squash operation is close to a horizontal or vertical shift.

Recursion Figure 6a depicts the representations of 2000 L9 expressions whose last applied operators (under the recursion) were subtraction (representations on horizontal line) and addition (vertical line), respectively. In Figure 6b we see that if these representations are the input of a new operation, they get projected onto the same diagonals (by the left and right part of the composition matrix, see equation 5) as the digit embeddings were. Although noise may be accumulated through the depth of the tree, this solution thus works recursively for longer expressions.



(a) Representations of L9 sentences (b) Projections of L9 representations

Figure 6: Representations of 2000 L9 expressions whose root operation was subtraction ($x \approx 1$, vertical line) or addition ($y \approx 1$, horizontal line). Right: how these would get projected by $\mathbf{W}_L$ (green) and $\mathbf{W}_R$ (purple) to serve as an input to a new computation.

5.2 Sequential model

Studying the internal structure of the sequential RNNs is more complicated, as the higher dimensionality of the models makes visual inspection impractical. Furthermore, the complex interaction between gates and hidden states in the GRU, as well as the fact that the networks are forced to incrementally process an input that is not strictly locally processable, makes it much less clear which computation is carried out at what point in time.

Diagnostic Classifiers To analyse the internal dynamics of the GRU, we propose a generic method that can be used to test and formulate hypotheses about solutions a model could be implementing on an algorithmic level [Marr, 1982]. Such hypotheses are formulated as sequences of targets for each

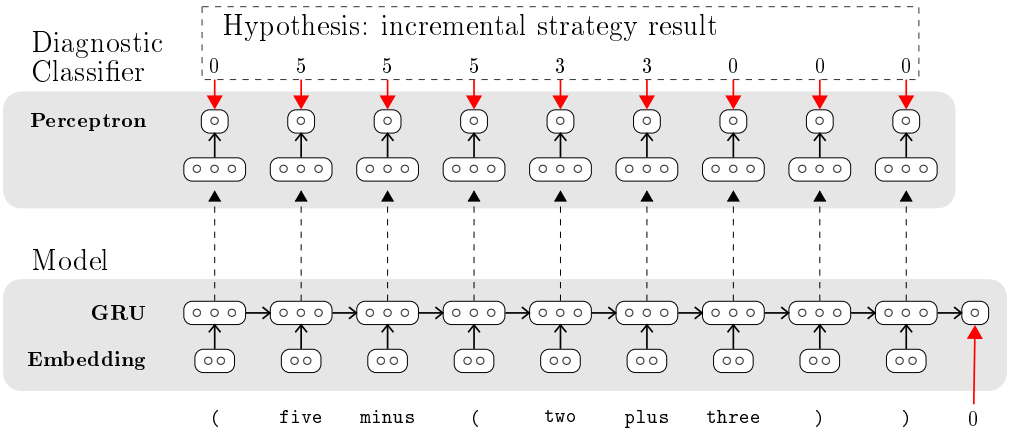
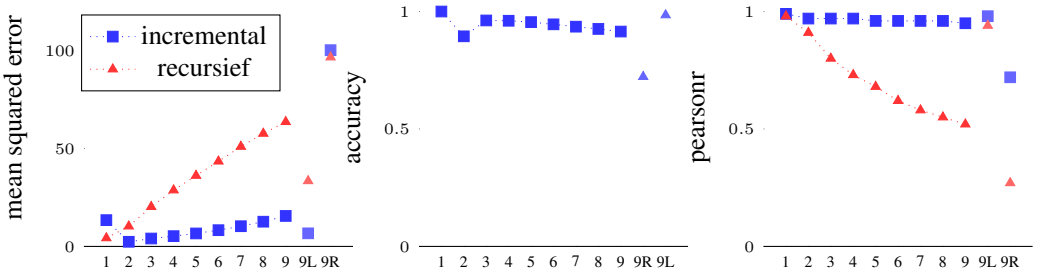


Figure 7: Testing a single hypothesis with a diagnostic classifier.

time step, and *diagnostic classifiers* are trained to predict such sequences from the RNNs hidden representations. The parameters of the original model are kept fixed during training.

For example, if the network were to follow the incremental strategy described in Figure 1b, the model should have a representation of the intermediate `result` at each point in time. If the sequence of intermediate results can be accurately predicted by a diagnostic classifier (see Figure 7), this supports the hypothesis that these values are in fact computed by the original model. As the diagnostic model should merely read out whether certain information is present in the hidden representations rather than perform complex computations itself, we use a simple linear model as diagnostic classifier. In the current paper, we test the hypothesis that the network follows either the incremental or the recursive strategy described in Figure 1. To this end, we train three diagnostic classifiers to predict the values of the intermediate results (`result` in Figure 1) and the variable `mode` used by the intermediate strategy to determine whether the next number should be added or subtracted.

Results We find that the values required for the incremental strategy (`mode` and `result`) can be more accurately predicted than the recursive intermediate strategy values (see Figure 8). From these findings it appears unlikely that the network implements a fully recursive strategy employing a stack of intermediate results. For the incremental strategy, on the other hand, the predictions are generally accurate, even for longer sentences. Notably, the diagnostic models exhibit a substantially lower accuracy for right-branching trees than for lengthy trees that are entirely left-branching. This is consistent with assumptions about the difficulty of employing information on larger size stacks as, contrary to right-branching trees, left-branching trees can be processed strictly locally. However, relatively high errors for sentences from L1 and L2 (for `result` and `mode` prediction, respectively) reveal that the network deviates from this strategy at least at some points.



(a) prediction of `result` (b) prediction of `mode` incremental (c) trajectory correlations

Figure 8: Results of diagnostic models for GRU on different subsets of languages

Trajectories A similar conclusion can be drawn from studying the predictions of the diagnostic classifiers. The predictions of the diagnostic classifiers on two randomly picked L9 sentences, along with their target trajectories as defined by the hypotheses, are depicted in Figure 9. These trajectories confirm that the line representing the incremental strategy is much better tracked than the recursive one (a correlation test over 5000 L9 sentences shows the same trend: Pearson's r = 0.52 and 0.95 for recursive and incremental, respectively).[4]

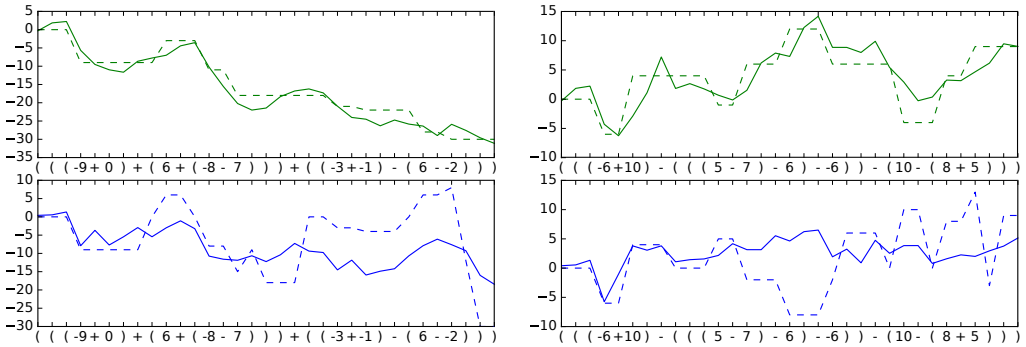


Figure 9: Trajectories of the incremental (green, upper) and recursive (blue, lower) classifier, along with their targets trajectories for the `result` values. Dashed lines show target trajectories.

6 Conclusions

In this paper we studied how recursive and recurrent neural networks process hierarchical structure, using a simple arithmetic language as a convenient toy task with unambiguous syntax and semantics and a limited vocabulary. We showed that recursive neural networks can learn to compute the meaning of arithmetic expressions and readily generalise to expressions longer than any seen in training. Learning was even successful when the representations were limited to two dimensions allowing a geometrical analysis of the solution found by the network. Understanding the organisation of the word embedding space and the nature of the projections encoded in the learned composition function, combined with vector addition and squashing, gave us a complete picture of the compositional semantics that the network has learned. This made clear that the network has found a near perfect approximation of a principled, recursive solution of the arithmetic semantics.

Still, the recursive neural network is a hybrid symbolic connectionist architecture, as the network architecture directly reflects the tree structure of the expression it must process, and this architecture is built up anew for each new expression using a symbolic control structure. This limits both the computational efficiency and the usefulness of this model for understanding how the human brain might process hierarchical structure. In this paper we therefore also analyse two recurrent neural network architectures: the classical Simple Recurrent Network and the recently popular Gated Recurrent Units. As it turns out, the latter can also learn to compute the meaning of arithmetic expressions and generalise to longer expressions than seen in training. Understanding how this network solves the task, however, is more difficult due to its higher dimensionality, recurrent connections and gating mechanism. To better understand what the network is doing we therefore developed an approach based on training diagnostic classifiers on the internal representations.

The qualitative and quantitative analysis of the results of a diagnostic classifier allows us to draw conclusions about possible strategies the network might be following. In particular, we find that the successful networks follow a strategy very similar to our hypothesised symbolic 'incremental strategy'. From this we learn something about how neural networks may process languages with a hierarchical compositional semantics and, perhaps more importantly, also provides an example of how we can 'open the black box' of the many successful deep learning models in natural language processing (and other domains), when visualisation alone is not sufficient.

[4]The plots in Figure 9 also point to possible further refinements of the hypothesis: where the predictions of the model change at every point in time, the targets value often remains the same for longer time spans, indicating that a hypothesis in which information is accumulated more gradually would give an even better fit.

References

Samuel R Bowman, Christopher D Manning, and Christopher Potts. Tree-Structured Composition in Neural Networks without Tree-Structured Architectures. In *Pre-Proceedings of Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches (CoCo @ NIPS2015)*, 2015a.

Samuel R Bowman, Christopher Potts, and Christopher D Manning. Recursive Neural Networks Can Learn Logical Semantics. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality (CVSC), Beijing, China, July 26-31, 2015*, pages 12–21, 2015b.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

François Chollet. Keras. *GitHub repository*, 2015.

Jeffrey L Elman. Finding Structure in Time. *Cognitive science*, 14(2):179–211, 1990.

Jerry A Fodor and Zenon W Pylyshyn. Connectionism and Cognitive Archirecture: A Critical Analysis. *Cognition*, 28(1-2):3–71, 1988.

Christoph Goller and Andreas Kuechler. Learning Task-dependent Distributed Representations by Backpropagation through Structure. In *International Conference on Neural Networks*, pages 347–352. IEEE, 1996.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching Machines to Read and Comprehend. In *Advances in Neural Information Processing Systems*, pages 1–13, 2015.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and Understanding Recurrent Networks. *International Conference on Learning Representations 2016*, pages 1–13, 2015.

Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13, 2014.

Phong Le and Willem Zuidema. Compositional Distributional Semantics with Long Short Term Memory. In *Prodeecings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM 2015)*, pages 10–19, 2015a.

Phong Le and Willem Zuidema. The Forest Convolutional Network : Compositional Distributional Semantics with a Neural Chart and without Binarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1155–1164, 2015b.

David Marr. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. *Phenomenology and the Cognitive Sciences*, 8(4):397, 1982.

James Martens. Generating Text with Recurrent Neural Networks. *Neural Networks*, 131(1):1017–1024, 2011.

Steven Pinker and Jacques Mehler. *Connections and symbols*, volume 43. Mit Press, 1990.

Paul Rodriguez. Simple Recurrent Networks Learn Context-free and Context-sensitive Languages by Counting. *Neural computation*, 13(9):2093–118, 2001.

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*, pages 1–9, 2010.

Sara Veldhoen. Semantic Adequacy of Compositional Distributed Representations. Master's thesis, University of Amsterdam, 2015.

Matthew D Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv*, 2012.