Visualisation and ‘diagnostic classifiers’ reveal how recurrent and recursive neural networks process hierarchical structure

Dieuwke Hupkes D.HUPKES@UVA.NL
Sara Veldhoen S.F.VELDHOEN@UVA.NL
Willem Zuidema ZUIDEMA@UVA.NL
ILLC, University of Amsterdam
P.O.Box 94242,
1090 CE Amsterdam, Netherlands

Abstract

We investigate how neural networks can learn and process languages with hierarchical, compositional semantics. To this end, we define the artificial task of processing nested arithmetic expressions, and study whether different types of neural networks can learn to compute their meaning. We find that *recursive* neural networks can implement a generalising solution to this problem, and we visualise this solution by breaking it up in three steps: project, sum and squash. As a next step, we investigate *recurrent* neural networks, and show that a gated recurrent unit, that processes its input incrementally, also performs very well on this task: the network learns to predict the outcome of the arithmetic expressions with high accuracy, although performance deteriorates somewhat with increasing length. To develop an understanding of what the recurrent network encodes, visualisation techniques alone do not suffice. Therefore, we develop an approach where we formulate and test multiple hypotheses on the information encoded and processed by the network. 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 that the networks follow a strategy similar to our hypothesised ‘cumulative strategy’, which explains the high accuracy of the network on novel expressions, the generalisation to longer expressions than seen in training, and the mild deterioration with increasing length. This is turn shows that diagnostic classifiers can be a useful technique for opening up the black box of neural networks. We argue that diagnostic classification, unlike most visualisation techniques, does scale up from small networks in a toy domain, to larger and deeper recurrent networks dealing with real-life data, and may therefore contribute to a better understanding of the internal dynamics of current state-of-the-art models in natural language processing.

1. Introduction

A key property of natural language is its hierarchical compositional semantics: the meaning of larger wholes depends not only on its words and subphrases, but also on the way they are combined. Subphrases, in turn, can also be composed of smaller subphrases, resulting in sometimes quite complex hierarchical structures. For example, consider the meaning of the noun phrase *the scientist who wrote the natural language research paper*, which is a combination of the meanings of *the scientist*, *wrote* and *natural language research paper*. The meaning of the latter phrase – a compound noun – is a combination of the meanings of *natural language* and *research paper*, which are combinations of the meanings of the individual words. Such hierarchical structures can be well represented by symbolic models (most famously, Chomsky, 1956; Montague, 1970), but if and how they can be represented by (artificial) *neural* models is an open question. This question has a long tradition in linguistics and computational neuroscience; recently, it has also received much attention from the field of natural language processing.

Although neural network models for language have been around since at least the 1980s, whether models are fundamentally capable of capturing the type of generalisations that are required for computing the meaning of such complex hierarchical structures is still largely unclear. From a theoretical perspective, even the simplest recurrent neural networks are known to be Turing complete (Siegelmann and Sontag, 1995) and many papers can be found that demonstrate the capability of (hand-crafted) recurrent networks to implement small context-free or even context sensitive languages (e.g. Gers and Schmidhuber, 2001; Rodriguez, 2001). However, both manually constructed and theoretical solutions rely on unrealistically high levels of precision and do therefore not scale up to real-life problems. Furthermore, the theoretical existence of solutions is not of much help if they are intractable in the vast parameter spaces of neural networks, or if they cannot be learned from (a finite amount of) data.

On the practical side, recent advances have shown that both recursive and recurrent artificial neural networks perform extremely well on natural language processing tasks such as machine translation (e.g. Sutskever et al., 2014; Bahdanau et al., 2015), sentiment analysis (e.g. Kalchbrenner et al., 2014; Socher et al., 2013; Le and Zuidema, 2015), semantic role labelling (e.g. Roth and Lapata, 2016) and recognizing textual entailment (e.g. Bowman et al., 2016). Although perfect performance on all of these tasks requires (implicit) knowledge of the underlying hierarchical structure and meaning of utterances of language, in practice it is unclear what sort of information the models are extracting from sentence representations to solve their task. Hence, also these models do not yet provide a convincing demonstration that artificial neural networks are adequately computing the meaning of sentences with hierarchical structure, nor are they convenient for improving our understanding of the means by which hierarchy and compositionality can be implemented in large collections of interconnected artificial neurons.

A deeper insight into the internal dynamics of artificial neural networks while they process hierarchical structures could prove valuable both from a theoretical and cognitive perspective, and is – considering the difficulty of searching through the vast parameter space of high dimensional neural networks – also interesting from an engineering point of view. However, such insight is not easily acquired by studying natural language with all its complexities directly. In this paper, we take a different approach. To analyse the mechanism of processing hierarchical compositional structures isolated from all other aspects involved in processing language, we study an artificial language – the language of arithmetic – of which both sentences, phrases and lexical items have clearly (and symbolically) defined meanings. First we investigate if and how *recursive* neural networks (Socher et al., 2010) can learn to correctly compute the meaning of sentences from this language, when their structure is given as part of the input. We present a thorough analysis of how they process them, using visualisations of the individual operations conducted by the network. We then extend our investigation to *recurrent* neural networks (Cho et al., 2014; Elman, 1990) which take a step forward to resembling the human brain, and constitute the current state-of-the-art in natural language processing.

We start with a definition of the arithmetic language that is the topic of our studies in Section 2; in Section 3, we define the models we are studying; Section 4 contains a description of our training methods, as well as a brief evaluation of the performance of the models on the defined task. The main contribution of this paper can be found in Section 5, where we present the analysis of the internal dynamics of the trained models.

The result of this study is a comprehensive analysis of the implementation of the composition function in the recursive neural network as well as an insightful interpretation of the symbolic strategy followed by the recurrent networks. Furthermore, we present a novel method to analyse high dimensional recurrent networks when visualisation techniques fall short. We use this method to gain more insight in our own trained networks, but also argue that it can be useful for analysis in many other cases where high dimensional recurrent neural networks are involved.

	sentences	meanings
L_1	{`minus ten, minus nine`, ..., `nine, ten`}	{`-10, -9`, ..., `9, 10`}
L_{m+n}	$\{(\, l_m\ \texttt{op}\ l_n\,) \mid l_m \in L_m,\ l_n \in L_n,\ \texttt{op} \in \{+, -\}\}$	$\langle l_m \rangle\ \texttt{op}\ \langle\, l_m \rangle$

Figure 1: Formal description of grammatical sentences s in the arithmetic language and their meanings $\langle s \rangle$.

L1	`one,    -three,    nine`
L2	`( five plus three )`
L3	`( ( two minus -three ) minus six ),    ( two minus ( -three minus six ) )`
L4	`( ( ( -two minus seven ) plus eight ) plus -ten)`

Table 1: Sentences from different subsets of the arithmetic language. Both numerals, operators and brackets are treated as words; words are represented by n-dimensional numerical vectors.

2. Arithmetic language

The vocabulary of the artificial language we consider consists of all integers in the range $\{-10, \ldots, 10\}$, the operators `+` and `-` and the brackets `(` and `)`. The grammatically correct phrases – i.e. sequences of words – in this *arithmetic language* comprise all grammatically correct, fully bracketed arithmetic expressions that can be formed with these symbols. The (compositional) meaning of an expression is the solution of the arithmetic expression that it spells out. For instance, the meaning of the phrase '`(10-(5+3))`' is 2.

We refer to expressions and sets of expressions by using the number of numeral words they contain (see Figure 1 for a formal definition). For instance, **L5** refers to all expressions with exactly 5 numerals and **l5** is an expression belonging to **L5**. Table 1 contains some example sentences of the arithmetic language, along with the name of the subset they may appear in.

The arithmetic language is specifically designed to allow us to study the mechanism of hierarchical compositionality in isolation, separate from other important aspects of natural language, such as structural and lexical ambiguity, irregular paradigms, multi-word units and idiomatic expressions. Furthermore, the symbolic nature of the arithmetic language allows us to formulate strategies to compute the meaning of expressions, which can be used to aid the dynamics of the internal dynamics of a network processing sentences. Before moving on to the description of our models and experiments, we will describe two possible strategies to incrementally solve arithmetic expressions.

Recursive strategy Perhaps the most obvious candidate for a symbolic strategy to compute the meaning of an arithmetic expression involves traversing through the expression, compute the outcome of all subtrees, and then cumulate the results at the end. To do this in an incremental fashion, the intermediate result of the computation of the current subtree should be pushed onto a stack – the `result_stack` – whenever a new, smaller subtree begins. At that point, also the operator that will later be used to integrate the outcome of the newly started subtree with its parent, should be stored on a stack. Because this stack determines whether the procedure is in *additive* or *subtractive mode*, we call it the `mode_stack`. Figure 2a contains a procedural description of this strategy; For a worked out example, see the upper part of Figure 3.

Cumulative strategy Alternatively, the meaning of a sentence from the arithmetic language can be computed by accumulating the numbers immediately at the moment they are encountered (see Figure 2b and the bottom part of Figure 3). This means that at any point during the computation a prediction of the solution of the expression is maintained. Consequently, this cumulative strategy does not require a stack with previous results, but it does require keeping track of previously seen

```
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 = apply(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) Cumulative strategy

Figure 2: Different symbolic strategies for incrementally solving arithmetic expressions. The function `apply`(*mode*, *result*, *symbol*) applies the operator specified by *mode* $(+, -)$ to the two numbers specified by *result* and *symbol*.

operators to decide whether the next number should be added or subtracted when a bracket closes (in Figure 2b captured by the variable `mode`).

Predictions following from strategies As illustrated in Figure 3, the two strategies do not only differ with respect to the computations they are executing, but also require different memory contents. Both the cumulative and the recursive strategy require a stack to store encountered operators. Consider for instance computing the outcomes of the expressions '`(8-((5-7)-2))`' and '`(8+((5-7)-2))`', for which information about the previous series of operators is required to understand if the `2` should be subtracted or added to the subtotal. In addition, the recursive strategy requires storage of the previously computed outcomes of subtrees. These differences result in different predictions about the sensitivity of the network to noise on the stack, implementation of the operator or depth of an expression, and hence in different predictions about the difficulty of processing certain structures under certain memory limitations and noise.

3. Models

We investigate two different types of artificial neural networks: *recursive* neural networks (TreeRNNs) and *recurrent* neural networks (RNNs). The former process sequences recursively, following a given syntactic structure, whereas the latter process sequences incrementally, reading them one word at the time.

3.1 TreeRNN

The TreeRNN (Goller and Kuechler, 1996; Socher et al., 2010) is a hybrid neural-symbolic model, in which a neural composition function is applied recursively, following a symbolic control structure. The key component of the TreeRNN is the composition function, that computes a representation of

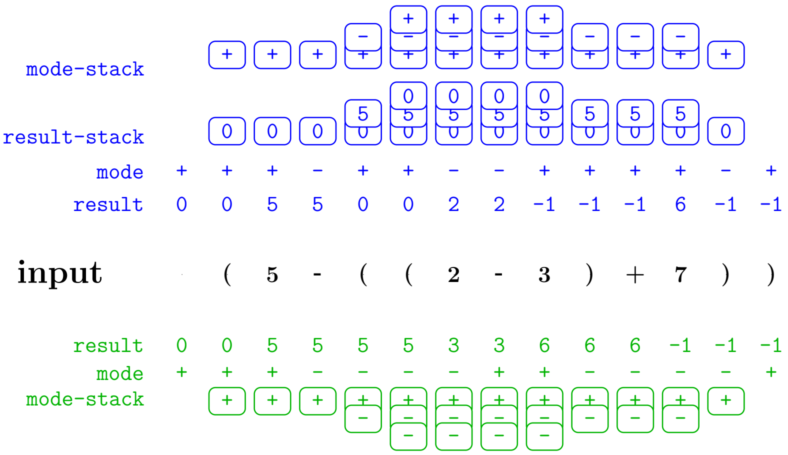

mode-stack
result-stack
mode + + + - + + - - + + + + - +
result 0 0 5 5 0 0 2 2 -1 -1 -1 6 -1 -1
input (5 - ((2 - 3) + 7))
result 0 0 5 5 5 5 3 3 6 6 6 -1 -1 -1
mode + + + - - - - + + - - - - +
mode-stack

Figure 3: Different strategies to compute the meaning of a sentence of the arithmetic language. Top (in blue): `mode-stack`, `result-stack` and current `mode` of the recursive strategy; Bottom (in green): cumulative `result`, `mode` and `mode-stack` of the cumulative strategy.

the composition of two or more words or phrases. In its most simple form this composition function is a single-layer feedforward network:

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

where $\mathbf{W} \in \mathbb{R}^{d \times nd}$ and $\mathbf{b} \in \mathbb{R}^d$ are learned. The input $[\mathbf{x_1}; \ldots; \mathbf{x_n}]$ is a concatenation of the input phrases or words $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. In case of the arithmetic language, there are three input vectors $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, representing a numeral, an operator and a numeral, respectively.

To compute the meaning of a sentence, the composition function is applied recursively, computing representations for all nodes in an externally provided phrase-structure tree in a bottom up fashion. Thus, to process the tree in Figure 4a, first the vectors for `two`, `plus`, and `three` are used, and Equation 1 is used to compute a vector for their combination. This new vector is then, in turn, combined with vectors for `five` and `minus`, to compute the meaning for the entire expression using again Equation 1. Note that the vectors representing words and those representing combinations of words have the same dimensionality.

As the syntactic structure of sentences in the arithmetic language is included in their definition, – and is thus unambiguous – training a TreeRNN to compute their meanings allows us to study how a composition function can be implemented in a neural model.

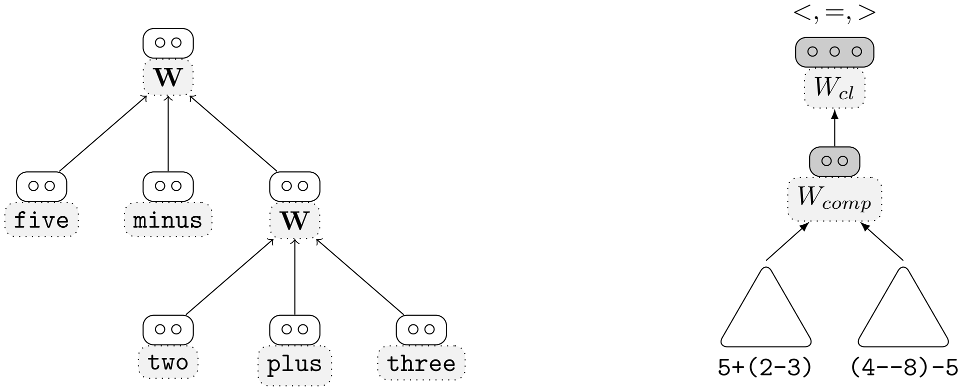
3.2 RNN

In addition to TreeRNNs, we study the dynamics of their sequential counterparts: RNNs, which will allow us to more easily scale up our findings and techniques to current state-of-the-art models. In particular, we consider two types of sequential networks: simple recurrent networks (SRNs), as originally introduced by Elman (1990), and their more recent extension Gated Recurrent Units (GRUs), defined by Cho et al. (2014).

SRN An SRN (see Figure 5a) consists of a single hidden layer $\mathbf{h}$ with a nonlinear activation function and a feedback connection that introduces a simple form of memory:

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

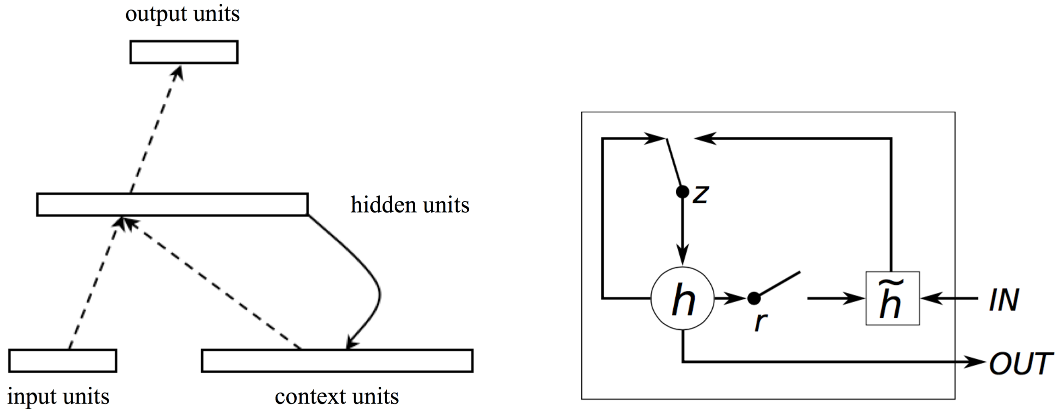
where the weight matrix $\mathbf{W} \in \mathbb{R}^{h \times d}$ connects the input $\mathbf{x} \in \mathbb{R}^d$ to the hidden layer $\mathbf{h} \in \mathbb{R}^h$ and the weight matrix $\mathbf{U} \in \mathbb{R}^{h \times h}$ stores the recurrent connections of the hidden layer to itself.



(a)

(b)

Figure 4: (a) A TreeRNN to compute the meaning of the sentence '`( 5 + ( 2 - 3 ) )`'. (b) Training setup of the TreeRNN. A comparison layer, on top of two TreeRNNs, is trained to predict whether left expression is smaller than, equal to or larger than the right one.



(a) SRN, picture from Elman (1990)

(b) GRU, picture from Chung et al. (2014)

Figure 5: Simple and gated recurrent neural network

GRU The GRU (see Figure 5b) 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 $\mathbf{r} \in \mathbb{R}^h$ is used to compute a candidate hidden state for the next time step $\tilde{\mathbf{h}}_t$:

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

where $\odot$ denotes the element wise product. The gate value $\mathbf{r}$ thus determines to what extent the hidden layer activation in the previous time step $\mathbf{h_{t-1}}$ should be taken into account, and is computed considering the previous activation as well as the current input, using weight matrices $\mathbf{W_r} \in \mathbb{R}^{h \times d}$, $\mathbf{U_r} \in \mathbb{R}^{h \times h}$ and bias $\mathbf{b_r} \in \mathbb{R}^h$:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r). \tag{4}$$

σ denotes the logistic sigmoid function. The update gate $\mathbf{z} \in \mathbb{R}^h$ then determines the proportion of a hidden unit's previous activation $\mathbf{h}_{t-1} \in \mathbb{R}^h$ that is retained, and how much it is updated with $\tilde{\mathbf{h}}_t$:

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t, \tag{5}$$

where $\mathbf{z}$, associated with its own weight matrices $\mathbf{W_z}$, $\mathbf{U_z}$ and bias $\mathbf{b_z}$, is computed analogously to the other gate:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{6}$$

All weight matrices $\mathbf{W} \in \mathbb{R}^{h \times d}$ thus connect the input $\mathbf{x} \in \mathbb{R}^d$ to a gate or the hidden layer, respectively, and the weight matrices $\mathbf{U} \in \mathbb{R}^{h \times h}$ connect the gates to the hidden layer and the hidden layer to itself.

4. Training and performance

We run a series of experiments training several models on an arbitrarily sampled set of expressions from **L1**, **L2**, **L4**, **L5** and **L7** (3000 expressions from each subset). The composition of lengths of the training set remains the same accross all experiments, but which exact expressions of these five lengths are chosen depends on the random seed during initialisation of the experiment. We set the input dimensionality of the word embeddings to 2. The word embeddings are randomly initialised within a small range, and are updated during training. All models are trained using backpropagation with minibatches of size 24.

We test on a large sample of expressions from **L1**, **L2**, ..., **L9**. All test sets but **L1** and **L2** contain expressions unseen during training, but testing also on expressions from **L3**, **L6**, **L8** and **L9** allows us to test also for generalisation to both longer and shorter unseen *structures*.

4.1 TreeRNN

The TreeRNN models are trained using a training regime inspired by Bowman et al. (2015b). In this regime, the composition function of the network is used to compute the meaning representations of two distinct sentences. These representations are then given as input to an additional neural classifier, that predicts the relation between the concatenated sentence representations (see Figure 4b). In our case, this relation is the (in)equality ('<', '=' or '>') that holds between their (numerical) solutions. The parameters (i.e. the composition matrix and the word embeddings) are updated through the optimisation of the performance of this classifier, using Adagrad (Zeiler, 2012) and a cross-entropy loss.

Results Out of the 8 TreeRNNs trained, 6 obtain a high classification accuracy between 0.98 and 0.99 on the test data (sentences up to length 9), the other 2 models perform at chance level. We will return to a more in depth analysis of the results later.

4.2 RNN

The sequential models are not provided with explicit structural information, but have access to the syntactic structure implicitly via the brackets, which are presented to the network as words. As a consequence, the sentences processed by the sequential models are substantially longer than the ones presented to the recursive network (e.g. 33 versus 17, respectively, for an **L9** sentence). All reported models are trained on an error signal backpropagated from a simple linear perceptron predicting the real-valued solution of the expressions, using Adam (Kingma and Ba, 2014) as optimiser (learning rate=0.001, β_1=0.9, β_2=0.999, ϵ=1e-08, decay=0.0) and mean squared error as loss function. The hidden layer dimensionality of all recurrent models is set to 15. For the implementation we use the Python library Keras (Chollet, 2015) with Theano (Theano Development Team, 2016) as backend.[1]

Results We train 20 SRN and 20 GRU model instances, that differ with respect to both weight initialisation and exact training instances. Of the 20 trained SRN models, three did not learn to capture any structural knowledge, reflected by a high error for short (but unseen) sentences with three numerals (**L3**). It is unclear to what extent the remaining 17 SRN models learned solutions incorporating the syntactic structure of sentences. Most GRU models, on the other hand, show a convincing ability to generalise, with a mean squared error that slowly increases with the length of the sentence. A summary of the results is plotted in Figure 6.

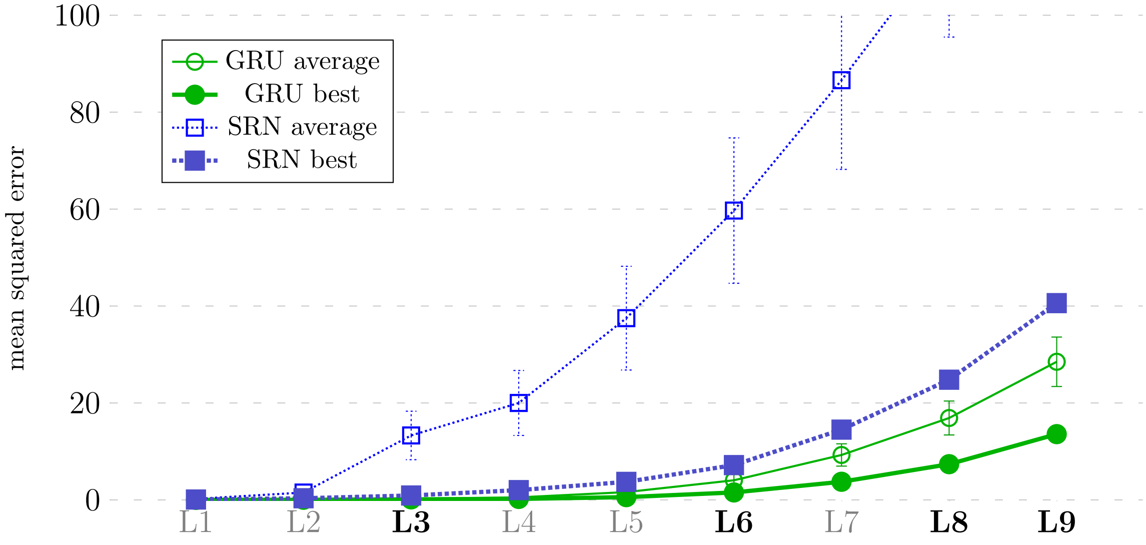


Figure 6: Average and best mean squared error for 20 GRU and 17 SRN models. Three of the trained SRN models that did not learn to capture any structural knowledge were excluded from the plot. Error bars indicate standard error. Sentences of lengths that were not included in the training set are bold-faced.

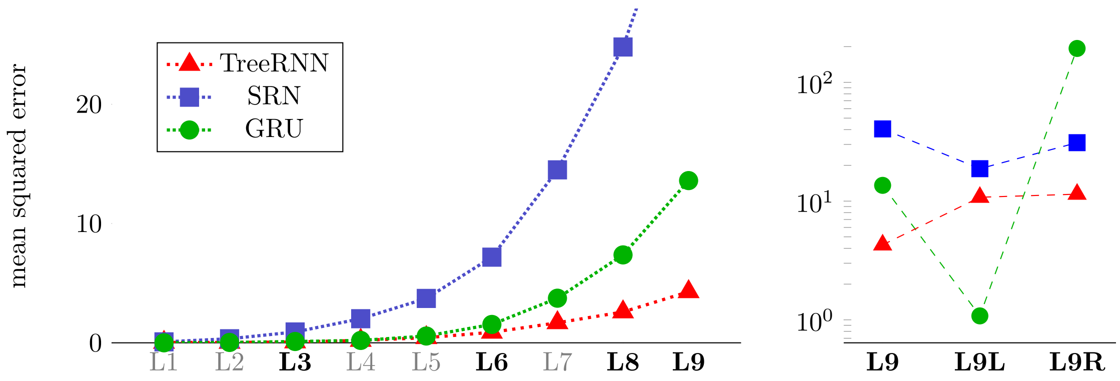
4.3 Discussion

In Figure 7 we plot the performances of the best TreeRNN, SRN and GRU models.[2] We find that the TreeRNN generalises adequately to longer sentences, evident from the smooth progression of the performance with respect to length (a criterion used by Bowman et al., 2015a). Although the generalisation capacity of the GRU (with $|h| = 15$) is weaker, also the GRU appears to have found a solution that is sensitive to the syntactic structure. From these results, it is unclear to what extent the best SRN model has learned a solution incorporating the syntactic structures of sentences.

A striking difference between the GRU and TreeRNN appears when considering their performance on sentences that are fully left- or right-branching (see Figure 7). While fully right-branching sentences are much harder to process for the GRU models, the performance of TreeRNNs suffers more from tree *depth*, regardless of the branching direction. We revisit this topic later.

[1] The Python package containing the source code of this project can be found at `https://github.com/dieuwkehupkes/processing_arithmetics`.

[2] To be able to compare the results of the recursive and recurrent neural networks, we trained a neural network with one hidden layer to predict the meaning of sentences from the representations generated by the best TreeRNN.



(a) (b)

Figure 7: (a) Mean squared error of the best performing SRN, GRU and TreeRNN models tested on sentences with a different number of numerals. (b) Mean squared error (log scale) of the best performing models tested on left- and right branching sentences containing 9 numerals (**L9L** and **L9R**, respectively). Sentence lengths not included in the training set are bold-faced.

5. Analysis

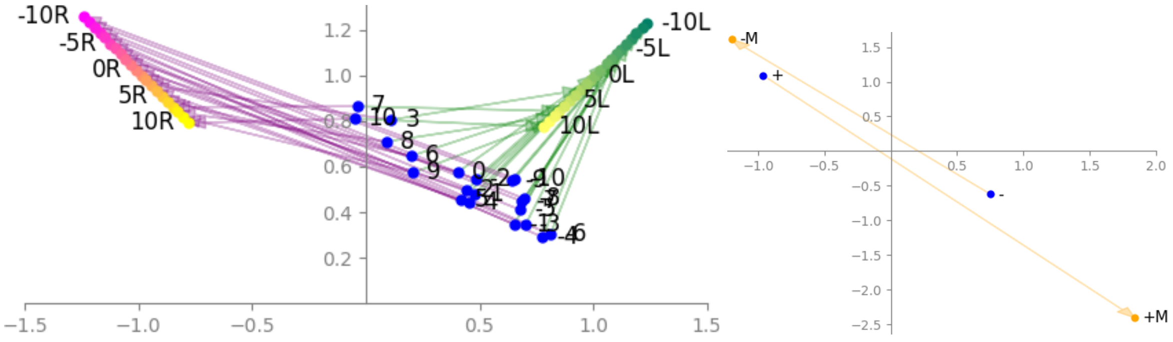
In the remainder of this paper, we present an analysis of the internal dynamics of the best performing models. We start by considering the TreeRNN, whose extremely low dimensionality allows us to inspect and understand the information encoded in the hidden layers and weight matrices. Subsequently, we 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

The recursive architecture of the TreeRNN forces the model to compute the meanings of the input sentences in a purely recursive fashion. Consequently, we do not have to discover *what* the model is doing, but only *how* this is implemented in the composition function. For this analysis, we use a method we call *project-sum-squash*, in which we separately consider the three operations executed by the composition matrix (see also Veldhoen, 2015). First, we consider how the three input components (numeral, operator, numeral) are *projected* by the composition matrix to different positions in the two dimensional state space. Next, the net input to the neuron's of the parent layer is effectively the *sum* of these three projections and the bias. Lastly, we study the effect of applying the activation function of the network, which *squashes* the net input back into the range $\{-1, 1\}$.

Project As the composition matrix $\mathbf{W}$ can be written as a concatenation $[\mathbf{W}_L; \mathbf{W}_M; \mathbf{W}_R]$ – where each $\mathbf{W}_i \in \mathbb{R}^{2\times 2}$ – we can consider the projections of the three input children individually. Figure 8a depicts the numeral embeddings and their projections, illustrating how the initially seemingly unorganised numerals embeddings are projected to organised, almost orthogonal subspaces, depending whether they are the left or right child. The projections of the plus and minus operator are roughly opposite to each other (see Figure 8b).

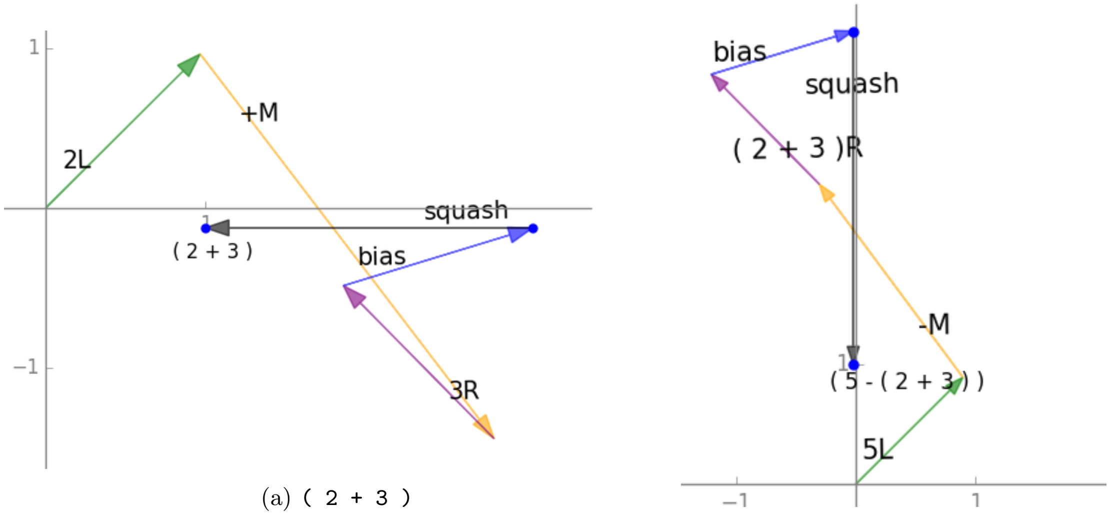
Sum After the representations are projected, they are summed and the bias term is added, as depicted by the coloured arrows in Figure 9. Due to the approximately orthogonal and relatively small subspaces the projections of the projected left and right representations inhabit, the result is a position close to one of the axes. The operator determines whether this is the x-axis (for +) or y-axis (for −).



(a)

(b)

Figure 8: Trained word embeddings and their projections. In (a) the projections of the seemingly unorganised numeral embeddings (blue) through $\mathbf{W}_L$ (as left child, green) and $\mathbf{W}_R$ (as right child, purple) reveal their organisation. In (b) we see the embeddings of the operators '`+`' and '`-`' and their projections after multiplication with $\mathbf{W}_M$ (orange).



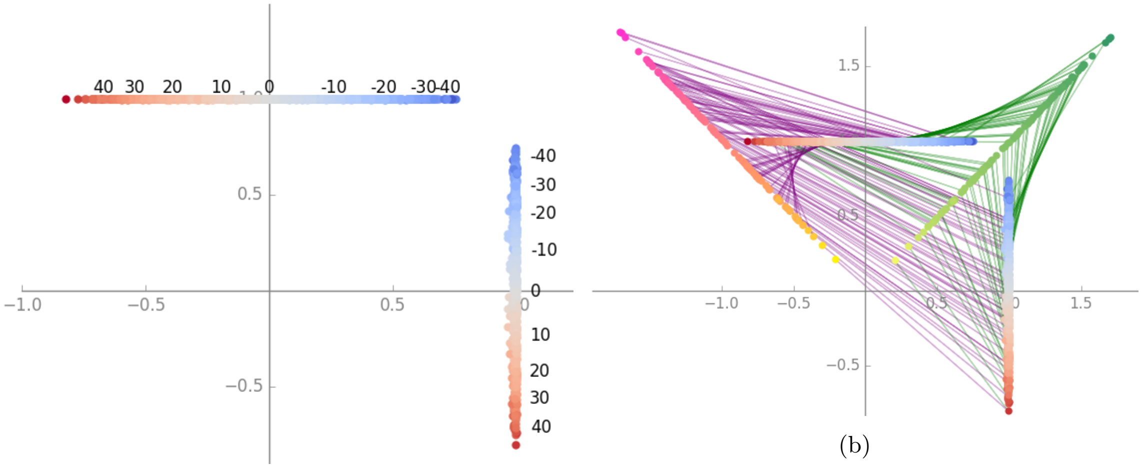
(a) `( 2 + 3 )`

(b) `( 5 - '( 2 + 3 )' )`

Figure 9: Summation of projections – left (green), middle (yellow) and right (purple) child – and the bias term (blue). Squash (tanh activation, grey) shifts the result back to the range $(-1, 1)$.

Squash In the last step, both coordinates of the representation are squashed into the range $(-1, 1)$ by the activation function `tanh`. After applying the composition matrix, the sum of the projected representations is already within this range for one of the dimensions and well above it for the other (which depends on the operator). The net effect of the squash function is therefore a horizontal or vertical shift. Figure 10a shows the representations of 2000 **L9** sentences, showing that this effect remains constant when the length of expressions grows.

Recursion In the *project* and *squash* steps it becomes clear that the initial word embeddings and the vectors representing larger phrases do not inhabit the same subspace. Where the word embeddings exist on a seemingly unorganised diagonal line (see Figure 8a), the larger phrase representations exist on horizontal ($y = 1$) and vertical lines ($x = 1$). However, Figure 10b illustrates that the left and right projections of both phrase and word representations do get projected onto the same orthogonal subspaces. This means the composition function can in theory be applied recursively for an indefinite number of steps.



(a)

(b)

Figure 10: (a) Representations of 2000 **L9** expressions whose root operation was addition ($x \approx 1$, vertical line) or subtraction ($y \approx 1$, horizontal line). (b) The projections of the same 2000 expressions as left and $\mathbf{W}_L$ (green) right $\mathbf{W}_R$ (purple) child when they serve as input to a new computation.

In summary, the project-sum-squash analysis nicely illustrates the process by which the TreeRNN recursively computes the compositional meaning of sentences in the arithmetic language.

5.2 RNN models

Although insightful for the extremely low dimensional recursive networks,the visualisation based project-sum-squash method becomes increasingly unpractical when the dimensionality of the networks increases. Additionally, the sequential structure of the RNN models and the complex interaction between gates and hidden states poses difficulties for the analysis of their internal dynamics. While the TreeRNN models process their input recursively by definition, the sequential models are forced to follow an incremental protocol to process the recursively structured sentences, which makes it much harder to discover which operation is carried out at which point in time. In other words, for the TreeRNN models we had to figure out *how* models are implementing the operations to compute the meaning of an expression, for the RNN models we also need to determine *what* these operations are. A sequential equivalent of the project-sum-squash method is thus difficult to realise, and for the analysis of the dynamics of our RNN models we have to resort to different methods.

In this section we first briefly discuss and test some previously proposed methods for visualising the internal state dynamics of neural networks, described in papers with an objective similar to ours (e.g. Karpathy et al., 2015; Li et al., 2016; Tang et al., 2017, all present detailed analyses of gated RNNs performing tasks involving compositionality). Subsequently, we discuss the shortcomings of these methods and propose an approach we call 'diagnostic classifiers' to test hypotheses on which strategy a trained neural network is effectively following. At the end of this section, we apply this method to interpret the inner workings of the best performing GRU model trained on the arithmetic language.

5.2.1 Visualising neural networks

Understanding the internal dynamics of recurrent neural networks is a topic that is receiving more and more attention. Most papers we are aware of concern visualisation of properties of internal states of RNNs. For example, Karpathy et al. (2015) and Li et al. (2016) study cell and gate activations under different conditions; Tang et al. (2017) plot distributions of cell activations and temporal traces of t-SNE cell vectors of a speech recognition RNN; and Strobelt et al. (2016) present an elaborate tool[3] that facilitates visual analysis of gated RNNs. We briefly discuss some of their methods and results and employ them to analyse our own models.

Individual cell dynamics A potential strategy to gain a better understanding of the internal dynamics of a recurrent neural network is to consider the interpretability of individual cells. By plotting individual cell activations, Karpathy et al. (2015) discovered several interpretable cells in a trained gated neural language model, including cells that keep track of the scope of quotes and cells that represent the length of the sentence. We here apply this approach to the best performing GRU model from our experiments. In Figure 11 we plot the hidden layer activations while the network is processing three different input sequences. Next to the activations, we put both the input sequence (starting from the top) as well as the `mode` of operation of the cumulative strategy in Figure 2 (i.e. '`+`' or '`-`') at each point in time. Under the hidden layer activations, we also plot the weights of the output layer reading out the meaning of the sentence, using the same colour scale as for the hidden layers (plotted at the right of the figure).

From this picture, several interesting observations can be made. Both the first cell (the first column from the left in all three graphs, black arrow) and the twelfth cell (the fourth column from the right, black arrow) show a sharp change in activation whenever the mode changes from '`+`' to '`-`'. The last layer of the network (bottom of the figure) indicates that the left most cell is negatively influencing the prediction of the solution of the expression, while the twelfth cell is hardly involved. The very last cell (red arrow) seems to respond to a minus in the input, but appears to have also other functions. The tenth cell (blue arrow) could potentially be representing the scope of a minus.

In summary, studying hidden layer activations is an interesting puzzle and can – especially for relatively low dimensional network such as ours – give pointers to which aspects should be studied in more depth. However, it is hard to draw definite (and quantitative) conclusions, and the usefulness of the method decreases with higher dimensionality of the networks.

Gate activation statistics In the same paper, Karpathy et al. (2015) study gate activations, focusing in particular on the fraction of time that gates spend being left- or right-saturated (activations less 0.1 than or more than 0.9, respectively). Such results are not easy to interpret, but might indicate what roles specific cells play in the information processing a network performs: are they mostly acting as a memory, in a feed forward fashion, or as standard recurrent cells without any additional form of memory? For example, a cell with a right saturated update gate remembers its previous activation, whereas a cell with a left-saturated update gate and a left-saturated reset gate ignores all previous activations and bases its value only on the next input.

In Figure 12 we plot the left- and right-saturation statistics for a GRU network for different lengths of expressions. As the dimensionality of our network is considerably lower than that of the network considered by Karpathy et al., we can easily visualise the gate saturation values for different sentence lengths in the same plot. We observe that most update gates are either on the x- or y-axis. Some cells (at the right of the picture) act as a memory a substantial amount of the time. A few cells show an interesting context dependency, spending an increasing fraction of the time being right-saturated. The reset gate saturation values show that several cells spend a considerable amount of time in 'feedforward-mode' (indicated by a high fraction of left-saturation). For one cell, the activation appears to be dependent on the length of the expression.

[3] LSTMVis, `http://lstm.seas.harvard.edu/`, is a visualisation tool that can be applied to gated RNNs, including the GRU model.

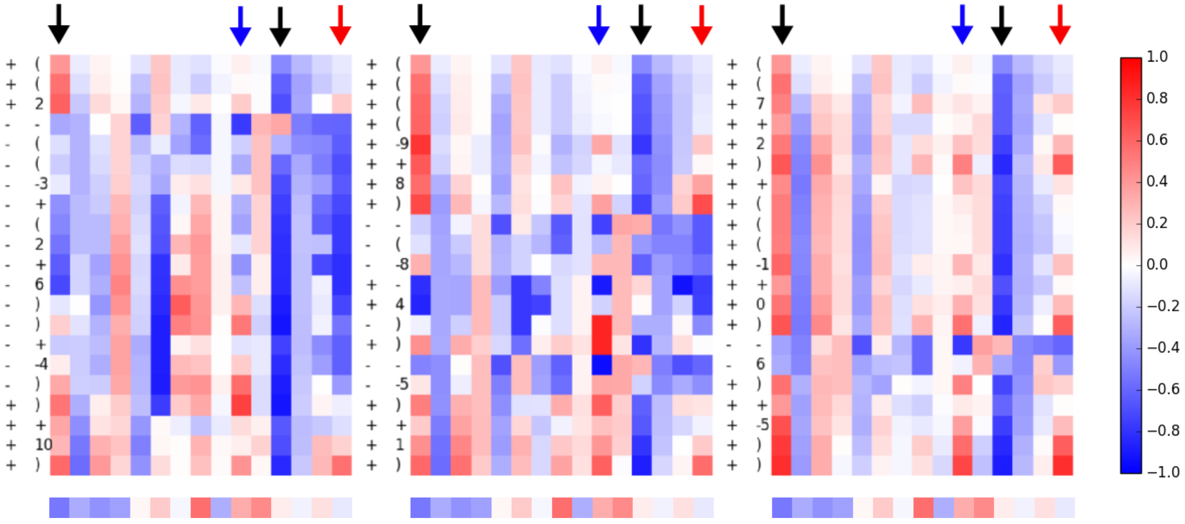
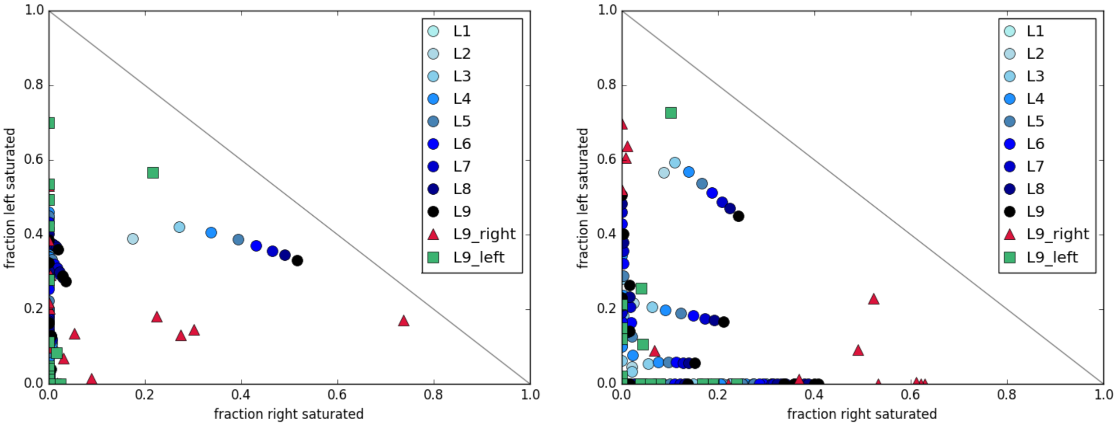


Figure 11: Hidden layer activations of a trained GRU network while processing different sequences. The input labels, along with the `mode` (addition/subtraction) at every point in time are printed left of the activation values. The activation values of cell 1 and 12 (black arrows) seem to be correlated with the `mode`, but it is is not easy to determine whether this value is in fact computed by the network. The 10th cell could be keeping track of the scope of a minus.



(a) Reset gate **r** (Equation 4)

(b) Update gate **z** (Equation 6)

Figure 12: Gate activation statistics of the update and reset gate of a GRU model while processing sentences from different lengths. Following Karpathy et al. (2015), each circle represents a gate in the GRU. Its position is determined by the fraction of the time it is left- (activation value less than 0.1) or right-saturated (activation value more than 0.9).

5.2.2 Diagnostic Classifiers

Although visualisation-based methods give intriguing clues, they do not result in a comprehensive view of the internal dynamics of a network. The potential conclusions that can be drawn from visual inspection concern only small parts of the networks overall behaviour and are often qualitative rather

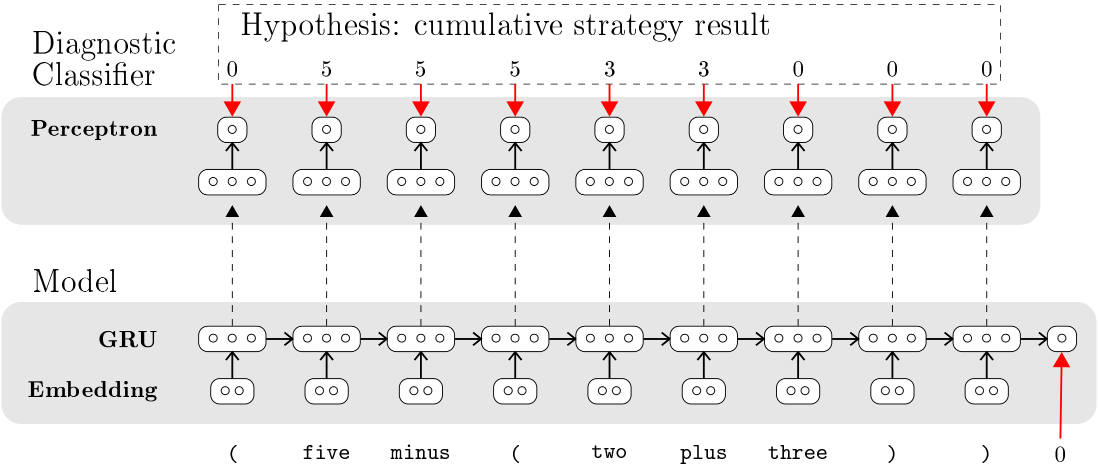


Figure 13: Testing a single hypothesis with a diagnostic classifier.

than quantitative, as manually analysing the behaviour of cells over a large number of examples is infeasible. Additionally, the described visualisation methods are restricted to finding functions or features that are encoded by one cell, while being insensitive to operations distributed over multiple cells, or cells that encode multiple features at the same time. Disentangling the behaviour of networks through visual inspection of activations is searching for a needle in a haystack.

We here develop an alternative approach that we call 'diagnostic classification'. Our approach is based on the idea that if a sequential model is computing certain information, or merely keeping track of it, it should be possible to extract this information from its internal state space. To test whether a network is computing or representing a certain variable or feature, we determine the sequence of values that this variable or feature takes at each step while processing a sentence. We then train an additional classifier – a *diagnostic classifier* – to predict this sequence of variable values (representing our hypothesis) from the sequence of hidden representations a trained network goes through while processing the input sentence. If the sequence of values can be predicted with a high accuracy by the diagnostic classifier, this indicates that the hypothesised information is indeed computed by the network. Conversely, low accuracy suggests this information is not represented in the hidden state. A sketch of the diagnostic classification method applied to test a hypothesis is depicted in Figure 13 (which we later discuss in more detail).

Diagnostic classification is a generic method that addresses most of the shortcomings we listed for existing methods. It can be used to quantitatively test hypotheses about neural networks that range from very simple to fully fledged (symbolic) strategy descriptions. This approach, which bears similarities with analysis methods presented by Adi et al. (2017) and Gelderloos and Chrupała (2016), can be used to quantitatively test for the existence of feature detectors such as the inside-quote detectors found by Karpathy et al. (2015), but also can be extended to test whether a network is computing the type of information needed for the symbolic strategies defined in Figure 2.

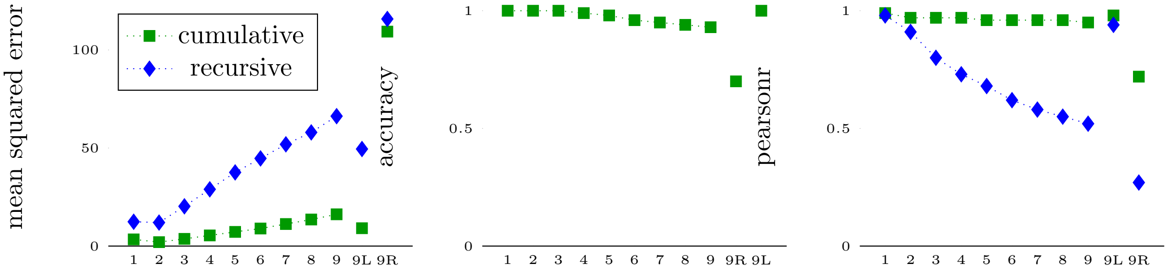
Feature testing Diagnostic classifiers can test whether a network has an internal representation of certain features of its input. For instance, to evaluate whether a network is keeping track of the length of a sentence (another example taken from Karpathy et al., 2015), we can simply train a diagnostic classifier to predict the value of this variable at each point in time from the hidden state of the network while processing a corpus of sentences. The accuracy of this diagnostic classifier will be high not only if there is a single cell acting as a length counter, but also when multiple cells are together encoding this information. Furthermore, the accuracy with which this classifier can predict the sentence length from the sequence of hidden states gives a quantitative measure of how well this information is kept track of. Analysing the accuracy of the classifier in more detail (i.e. inspecting at which points it fails to correctly predict the sentence length) or looking at its weights can provide more insight in what the network is doing.

Symbolic strategies In a similar fashion, diagnostic classifiers can be used to probe the strategy networks could be implementing on an algorithmic level (Marr, 1982), provided such strategies can be translated into sequences of targets for each time step. The cumulative and recursive strategy we defined in Figure 2b and Figure 2a, respectively, result in very different predictions about the intermediate results stored (and computed) during processing a sequence. For instance, after seeing the word '`3`' in the sequence '`(5 - ( ( 2 - 3 ) + 7 ) )`', the recursive strategy should have a representation of the value within the current brackets (which is -1), whereas the cumulative strategy should maintain a representation of the value of the expression up to that point (6). Figure 13 shows the setup for training a diagnostic classifier to predict the intermediate results for the cumulative strategy.

5.2.3 Applying Diagnostic Classifiers

To test whether our trained networks are following either the cumulative or recursive strategy, we train diagnostic classifiers to predict the sequences of intermediate results of both these strategies, as well as the variable `mode` used by the cumulative strategy to determine whether the next number should be added or subtracted. 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.

Strategy results We find that the values required for the cumulative strategy (`mode` and `result`) can be more accurately predicted than the intermediate recursive strategy values (see Figure 14a and 14b). From these findings it appears unlikely that the network implements a fully recursive strategy employing a number stack of intermediate results. For the cumulative strategy the predictions are generally accurate, even for longer sentences. The same is true for the `mode` of subtraction of the cumulative strategy (see Figure 14b), which can be predicted almost perfectly for sentences up until length 5 (with an accuracies in the range of 0.98 – 1.0), but is also accurately kept track of for longer sequences (an accuracy 0.93 for **L9** sentences). However, the fit with the cumulative strategy is not perfect: the diagnostic classifiers trained to test the cumulative hypothesis perform excellently for left-branching sentences, but show low accuracy for right-branching sentences. This is inconsistent with the symbolic description of the cumulative strategy, where the stack is crucial for left-branching sentences, but not relevant at all for right-branching sentences. We revisit this issue in the discussion part of this section.



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

Figure 14: Results of diagnostic models for a GRU model on different subsets of languages.

Plotting trajectories We can not only use diagnostic classifiers to evaluate the overall match with a specific hypotheses, we can also track the fit of our predictions over time, by comparing the trajectories of predicted variables with the trajectories of observed variables while the networks process different sentences. In Figure 15, the predictions of the diagnostic classifiers on two randomly picked L9 sentences are depicted, along with their target trajectories as defined by the hypotheses. These trajectories confirm that the line representing the cumulative strategy is much better predicted 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 cumulative, respectively. Figure 14c shows the trajectory correlations for test sentences of different lengths.

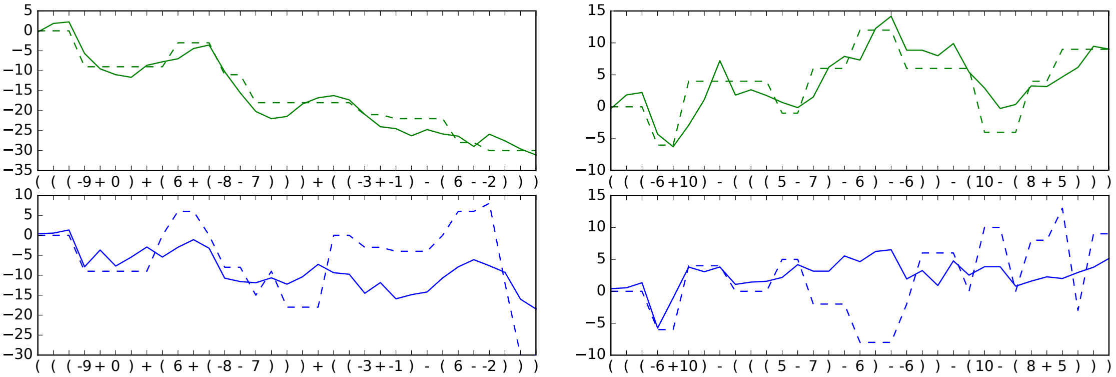


Figure 15: Trajectories of the cumulative (green, upper) and recursive (blue, lower) classifier, along with their targets trajectories for the `result` values. Dashed lines show target trajectories.

However, we also observe an important qualitative difference between the diagnostic classifier trajectories and the target values: the diagnostic classifier trajectories are smooth, changing value at every point in time, where the target trajectories are jumpy and often stay on the same value for longer time spans. This once more indicates that a refinement of the symbolic cumulative hypothesis, in which information is integrated more gradually, would be more suitable for a network like this.

5.2.4 Discussion

Using diagnostic classifiers, we have been able to analyse the internal dynamics of the GRU network in more detail than ever before. This allows us to conclude that the GRU is achieving its suprisingly good accuracy and generalisation behaviour, by following a strategy that roughly approximates the symbolic 'cumulative strategy'. At the same time, it also provides us with evidence that the network's strategy does deviate from the cumulative strategy in details, and raises new questions.

Left- and right-branching expressions One of such questions concerns the difference between left- and right-branching expressions, and the fact that the GRU's behavior doesn't match the cumulative strategy's predictions very well for right-branching expressions. Understanding the reasons for this mismatch requires more research into the type of structures and operators that are difficult for the network or a more elaborate study of the trajectories of the trained diagnostic classifier for left- and right-branching sentences.

Diagnosing gates Another open question concerns the exact role of the gating mechanism in computing the outcomes of arithmetic expressions. While we in this study focused on understanding the information in de hidden layer, diagnostic classifiers can certainly also be used to diagnose the role of *gates*. Their use there does requires some extra care: where the hidden layer is directly connected to the output layer and it thus is reasonable to assume that computed information can be be predicted with a linear classifier, the information stored in the gates is used by the network only after passing through at least one non-linearity. This suggests that diagnostic classifier trained to diagnose gates should make use of a more complex architecture that can handle non-linearities, but this bears the risk that the classifier infers features that are not actually used by the network, but can be inferred as an artefact of the projection of the low dimensional word embeddings in a high dimensional space.

More realistic tasks We have demonstrated that diagnostic classifiers can successfully extract interesting properties of low dimensional networks trained on an idealised task that would have been difficult to find otherwise. However, we have not yet experimented with deeper and higher dimensional networks and real world tasks. An exciting next step is using diagnostic classification for models trained on tasks such as language modelling, machine translation or semantic role labelling. Arguably, diagnostic classification could prove very useful for understanding the high dimensional networks typically used in these domains, for which visualisation is even more cumbersome.

6. Conclusions

In this paper we studied how recursive and recurrent neural network process hierarchical structures, using an arithmetic language as a convenient, idealised 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 geometric 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 neural-symbolic architecture, as the network architecture directly reflects the tree structure of the expressions it processes, 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 analyses of the results of a diagnostic classifier allow 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 ‘cumulative strategy’. From this we learn something about how neural networks may process languages with a hierarchical compositional semantics and, perhaps more importantly, also provide 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.

Acknowledgements

This paper is an extended version of Veldhoen et al. (2016), presented at a workshop at NIPS 2016. DH and WZ are funded by the Gravitation Program 'Language in Interaction' of the Netherlands Organization for Scientific Research.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *International Conference on Learning Representations (ICLR)*, 2017.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd Conference on Learning Representations (ICLR2015)*, 2015.

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.

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 ACL*, 2016.

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.

Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

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

Lieke Gelderloos and Grzegorz Chrupała. From phonemes to images: levels of representation in a recurrent neural model of visually-grounded language learning. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1309–1319, 2016.

Felix A Gers and Jürgen Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *Neural Networks, IEEE Transactions on*, 12(6):1333–1340, 2001.

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.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, page 655–665, 2014.

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. 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, 2015.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. In *Proceedings of NAACL-HLT*, pages 681–691, 2016.

David Marr. Vision: A computational investigation into the human representation and processing of visual information. *Phenomenology and the Cognitive Sciences*, 8(4):397, 1982.

Richard Montague. Universal grammar. *Theoria*, 36(3):373–398, 1970.

Paul Rodriguez. Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural computation*, 13(9):2093–118, 2001.

Michael Roth and Mirella Lapata. Neural semantic role labeling with dependency path embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1192–1202. Association for Computational Linguistics, 2016.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

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.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M Rush. Visual analysis of hidden state dynamics in recurrent neural networks. *arXiv preprint arXiv:1606.07461*, 2016.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Zhiyuan Tang, Ying Shi, Dong Wang, Yang Feng, and Shiyue Zhang. Memory visualization for gated recurrent neural networks in speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 2736–2740. IEEE, 2017.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Sara Veldhoen. Semantic adequacy of compositional distributed representations. Master's thesis, University of Amsterdam, 2015.

Sara Veldhoen, Dieuwke Hupkes, and Willem Zuidema. Diagnostic classifiers: Revealing how neural networks process hierarchical structure. In *Pre-Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches (CoCo @ NIPS 2016)*, 2016.

Matthew D Zeiler. ADADELTA: An adaptive learning rate method. *arXiv:1212.5701*, 2012.