Diagnostic classification and symbolic guidance to understand and improve recurrent neural networks

Hupkes, D.; Zuidema, W.

Publication date
2017

Document Version
Author accepted manuscript

Diagnostic classification and symbolic guidance to understand and improve recurrent neural networks

Dieuwke Hupkes
University of Amsterdam
Amsterdam, 1098 XX
dieuwkehupkes@gmail.com

Willem Zuidema
University of Amsterdam
Amsterdam, 1098 XX
w.h.zuidema@uva.nl

Abstract

This paper describes a search through a variety of methods to inspect and understand the internal dynamics of gated recurrent neural networks, using a task focusing on a key feature of language: hierarchical compositionality of meaning. We study both gated recurrent units and long short term memory networks with a technique called *diagnostic classification*, in which a simple neural meta-model is trained to qualitatively evaluate hypotheses about the information that is encoded in the hidden state of a trained network. Using this method, we analyse the hidden layer activations, but also the gates of trained networks. We explore the potential limits of diagnostic classification using not only the accuracy of the resulting diagnostic classifiers, but also studying their weights to understand where and how information is encoded. As a result, we develop a detailed understanding of the computations implemented by the networks to execute their task, which we then utilise to improve their performance with a new training regime we call *symbolic guidance*. This regime uses symbolically generated targets that are specifically designed to leverage our knowledge about the inherent bias of the networks and leads to a clear improvement against the best previous model.

1 Introduction

In many subfields of natural language processing, the current state-of-the-art models are all based on artificial neural network architectures [e.g. Bahdanau et al., 2015, Sutskever et al., 2014]. Until recently, the research leading to these successes primarily involved developing increasingly sophisticated architectures and training methods, often also resulting in large increases in the number of parameters. Our understanding of what the resulting models actually encode, however, remains poor. Opening the black box of these models is a crucial topic of research, both for the usefulness of these models as models of cognition, and for developing new training methods that more appropriately address their inherent biases.

In this paper, we present an exploration of methods to increase our understanding of neural models, focusing on a key linguistic topic: (hierarchical) compositionality of meaning. As such, this paper can be seen as a joint effort of developing and evaluating methods to inspect the dynamics of recurrent neural networks, while increasing our understanding of how they can implement hierarchical compositionality. To do so, we will employ the method of *diagnostic classification* [Hupkes et al., 2017], in which hypotheses about what is encoded in the hidden state of a neural network are tested by training additional neural classifiers to predict features following from these hypotheses. We profit from our increased understanding by using *symbolic guidance*, a new approach that exploits the most fruitful hypotheses according to diagnostic classification to guide the network in the learning process.

We first describe our initial experiments in Section 2. In Section 3, we present a thorough analysis of the internal dynamics of the trained networks. In Section 4, we present *symbolic guidance*, our novel

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

training method. Throughout the paper, we provide several suggestions for further research, and we finish with a discussion on what our search through methods of analysis has brought us by focusing on the question central to this workshop: *What now?*

2 Hierarchical compositionality in recurrent neural networks

We work with the toy task of processing sentences from a lexical *arithmetic language*, consisting of spelled out, nested, arithmetic expressions [for more details, see Veldhoen et al., 2016]. These sentences comprise sequences of brackets, operators (`+` and `-`), and digits (`{-19,...,19}`), that are all represented by 2-dimensional word embeddings. The deep hierarchical structure and symbolic nature of this task allows us to isolate the aspect of hierarchical compositionality, while providing us with the luxury of formulating clear symbolic hypotheses about how they may be processed, which is usually not possible when working with natural data. A formal description of the grammatical sentences of this language can be found in Figure 1.

	Form	*Meaning*
L_1	{minus ten, minus nine, …, nine, ten}	{-10, -9, …, 9, 10}
L_k	$\{(\, l_m \, x \, l_n \,) \mid l_m \in L_m,\ l_n \in L_n,\ x \in \{+, -\},\ m + n = k\}$	$\langle l_m \rangle \, x \, \langle \, l_m \rangle$

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

In a first series of experiments, we assess to what extent gated recurrent networks can learn to correctly process the deep hierarchical structure of sentences from the arithmetic language, without being provided any explicit feedback on this structure.[1] We now first provide a global explanation of the workings of the two models that are our focus of study: Long Short Term Memory networks [LSTMs, Hochreiter and Schmidhuber, 1997], and Gated Recurrent Units [GRUs, Cho et al., 2014].

2.1 Models

Gated neural networks are an extension of simple recurrent neural networks [SRNs, Elman, 1990], proposed to mitigate vanishing or exploding gradient problems that arise when recurrent networks are trained with backpropagation through time [Hochreiter and Schmidhuber, 1997]. In this paper, we focus on two common versions of recurrent neural networks: LSTMs and GRUs.

LSTM An LSTM unit has three gates, that together modulate the recurrent information flow in the network. The gate values are computed considering the previous hidden layer activation and the current input:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \quad (1)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \quad (2)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \quad (3)$$

where σ denotes the logistic sigmoid function. The forget gate $\mathbf{f_t} \in \mathbb{R}^h$ and the input gate $\mathbf{i_t} \in \mathbb{R}^h$ together determine to what extent the content of the memory $\mathbf{c_t} \in \mathbb{R}^h$ cell is updated:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \quad (4)$$

where $\odot$ denotes the element wise product. The forget gate regulates how much of its previous activation is memorised, whereas the input gate determines how much of the recurrent activation is taken into account. With a completely closed forget gate ($\mathbf{f_t} = 0$) and a completely open input gate ($\mathbf{i_t} = 1$) , the activation of the memory cell matches the standard SRN activation; closing the input gate and opening the forget gate has as result that the previous memory activation is retained. The final hidden layer activation is modulated by the output gate:

$$\mathbf{h}_t = \mathbf{o_t} \odot \tanh(\mathbf{c}_t) \quad (5)$$

The output gate can thus completely reset the hidden state of the network if it is set to 0, while setting it to 1 results in merely copying the memory content.

[1] The network is trained using a mean squared error loss of a linear output unit against the meaning of the expression (i.e. its outcome) as target.

GRU Contrary to an LSTM unit, a GRU has only 2 gates, that are computed analogously to the LSTM gates, but have different functions.

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \quad (6)$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \quad (7)$$

The *reset* gate $\mathbf{r_t} \in \mathbb{R}^h$ is used to compute the candidate hidden layer activation for the next time step, and thus determines to what extent the candidate activation depends on the input and the previous hidden layer activation.

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}_h(\mathbf{r} \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \quad (8)$$

On the one extreme, the previous activation is not considered at all, and the candidate activation matches the feedforward activation considering the input ($\mathbf{r_t} = 0$). When $\mathbf{r_t} = 1$, on the other hand, the candidate hidden layer activation is equal to the activation of an SRN. The update gate $\mathbf{z_t} \in \mathbb{R}^h$ 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 $\tilde{\mathbf{h}}_t$:

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t, \quad (9)$$

When $\mathbf{z_t} \approx 1$ the unit functions as a memory, keeping its previous activation. For small $\mathbf{z_t}$, the unit is updated to match the candidate activation.

2.2 Results

We train 20 GRU and 20 LSTM model instances with one hidden layer with 15 hidden units and one output unit to predict the outcome of an arbitrarily sampled subset of expressions from **L1**, **L2**, **L4**, **L5** and **L7** (3000 expressions from each subset). Different runs differ with respect to weight intialisation and the exact sentences in the training set. Word embeddings are updated during training.

We test all models on a large sample of expressions from {**L1**, ..., **L9**}. In Figure 2, we plot a summary of the results. All models perform well above an SRN baseline and learn to generalise to both longer and shorter unseen sequences, as demonstrated by the smooth curve of the models' performance. The LSTMs perform better on average, while the performance of the best GRU and best LSTM model (decided by $L9$ performance) is comparable.

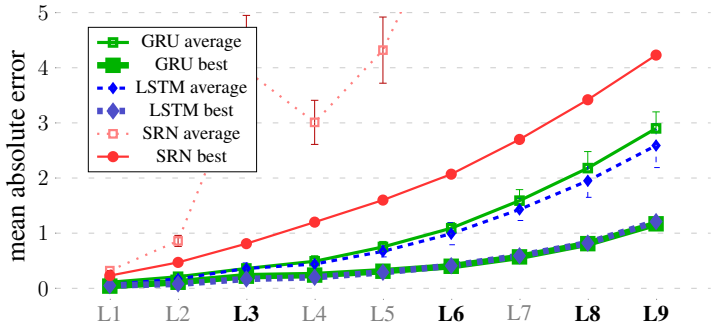


Figure 2: Mean absolute error of 20 runs of training for GRU and LSTM models.

3 Analysis

The initial results provide a convincing demonstration that both GRU and LSTM models have learned to interpret – to some extent – the compositional structure of expressions from the arithmetic language, and can use this to compute their meaning. We now proceed with analysing their internal behaviour, aiming to discover *how* they do this, and how they can be improved.

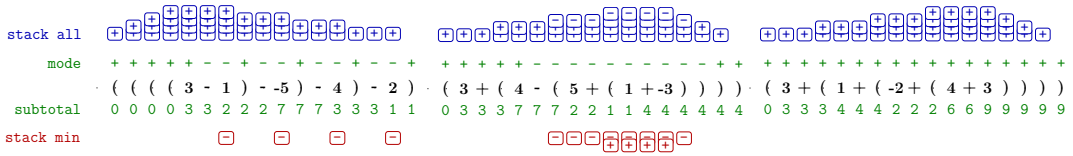
3.1 Diagnostic classification

In our experiments we use a technique called diagnostic classification Hupkes et al. [2017]. Diagnostic classification is based on the idea that if a model is using certain information during processing, it

should be possible to predict this information from its (hidden) state space, using a relatively simple readout model (e.g., a linear classifier). To test whether networks are computing or representing a certain variable or feature, an additional (neural) model is trained to predict the value of this variable or feature at each point in time from the states a network goes through while processing sentences. The accuracy of this model (the diagnostic classifier) is then taken as a qualitative measure of the extent to which this information is computed by the original model.

3.2 Symbolic strategy

Veldhoen et al. find that the models they train to compute the meaning of expressions of the arithmetic language follow a *cumulative strategy*, in which digits are integrated at the moment they are encountered during processing.[2] Two important components of this strategy are the `subtotal` – which represents the intermediate prediction of the outcome of the expression according to the cumulative strategy – and the operator `mode`, used to integrate the next digit with the `subtotal`. Figure 3 contains some examples of expressions and the corresponding sequences of values for `mode` and `subtotal`.



(a) Left-branching, minus (b) Left-branching, mixed (c) Right-branching, plus

Figure 3: Three expressions and the values of `mode` and `subtotal` during processing, plus two different examples of stacks used to compute the current `mode`.

Operator mode A possible way to compute the `mode` is to store the current `mode` on a stack whenever an opening bracket is encountered, and pop the next `mode` from the stack whenever a bracket closes (upper blue stack in Figure 3). Although elegant for its simplicity, this solution is not consistent with the apparent ease of processing left-branching expressions, which in the described strategy require heavy employment of the stack. The more difficult right-branching expressions instead, lead to accessing the stack only when all computation is already finished (e.g. Figure 3b and 3c). An alternative possibility is to begin to employ the stack only when brackets are preceded by a minus (lower, red stack in Figure 3). The stack for the latter strategy is shorter, and remains primarily empty for left-branching expressions (see Figure 3a).

Diagnostic Classifiers We train diagnostic classifiers to confirm that the networks are in fact integrating digits with the `subtotal` in a cumulative fashion, and find that indeed even for long sequences both the operator `mode` and the `subtotal` of the cumulative strategy are predicted from the hidden states with a very high accuracy for both GRU and LSTM models (not plotted). But how is this operator mode computed? One feature that distinguishes the full-stack computation of the operator mode from the one where the stack is only employed after a minus (upper blue and bottom red stack in Figure 3, respectively), is that the latter requires keeping track of the *scope* of minus operators. We train diagnostic classifiers to test if the hidden state encodes the property of being within the scope of at least one minus (we call this feature `minus_scope1+`), and how many closing brackets should still be seen to close this scope (`close_minus_scope1+`). Consider Figure 4 for a worked out example of an expression and the values these variable take as the expression progresses.

In Figure 5a, we see that virtually all networks maintain a representation of `minus_scope1+` during processing. Furthermore, the error on the prediction of the number of closing brackets that need to be encountered to close the scope of the minuses is very low (Figure 5b), indicating that `minus_scope1+` is indeed a salient feature for the network. Additional diagnostic classifiers (results not plotted), show that also the scope of minuses of deeper levels (minuses that are within the scope of another minus, such as `minus_scope2+` in Figure 4), can be inferred from the hidden states of the networks.

[2] An alternative strategy is be to follow a *recursive* algorithm, storing the outcomes of subtrees on a stack and integrating them only at the end of the expression when all brackets are closed.

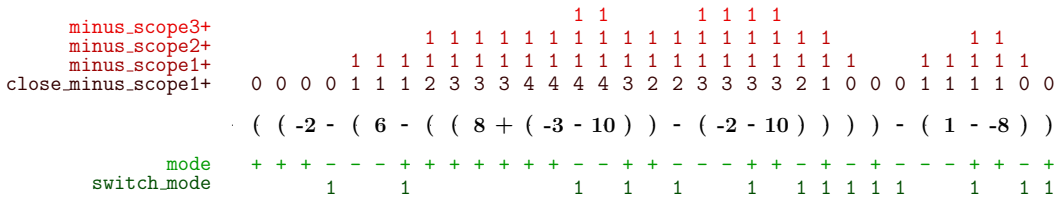


Figure 4: Example sentence plus the values of different features during processing. For clarity reasons, zero values of binary features are not printed.

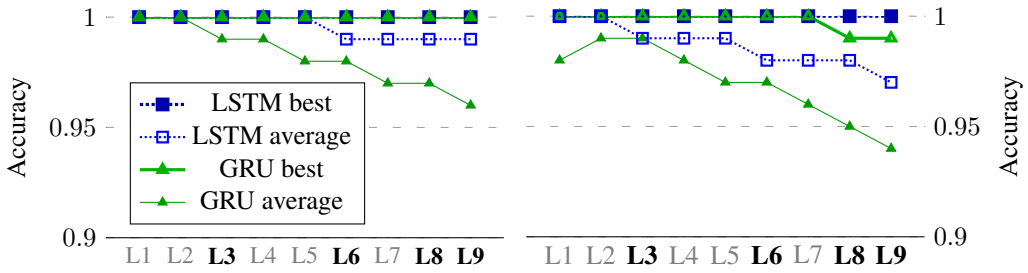


Figure 5: Binary accuracy of diagnostic classifiers trained on predicting the features `minus_scope1+` (left) and `close_minus_scope1+` (right) from the hidden layer activations of trained models.

3.3 Utilising Diagnostic Classifiers

Aside from providing a way to quantitatively determine whether information is encoded in the hidden states of a network, diagnostic classifiers can be also used to inspect where and how this information is represented. There are many possible ways to do so, we present one case study that revolves around understanding how information and processes are distributed and located inside a neural network, an aspect which may play a role when one wants to re-utilise learned functions and address the problem of catastrophic forgetting [Goodfellow et al., 2013, McCloskey and Cohen, 1989].

In Figure 6 we plot the values of the weights of the diagnostic classifiers trained to predict the variable `minus_scope1+` for the best GRU and LSTM models. Looking at these weights, the representation of this variable seems to be distributed over several different units, that are connected to the diagnostic classifier with a high weight. To measure the contributions of different units of the network to the final accuracy, we test how well individual weights of the diagnostic classifier predict the feature on their own, when all other weights of the classifier are set to 0. The result of this masking experiment tells a different story than the weights visualisation: in both the LSTM and GRU model, there is one single unit that can predict `minus_scope1+` with almost perfect accuracy on its own, without using any of the other units (Figure 7a and Figure 7b, colours match the weight visualisation in Figure 6). Whether this is an artefact of the training regime leaving traces of previous attempts to solve the problem, a way to ensure robustness of the network, or simply a consequence of the fact that different functions implemented by the network are related, it is certainly worth exploring further.

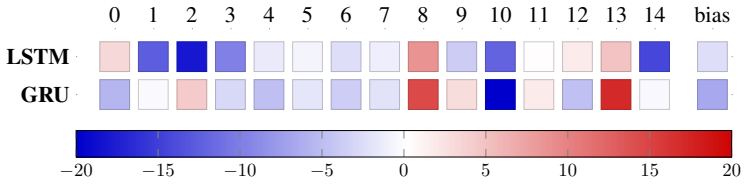
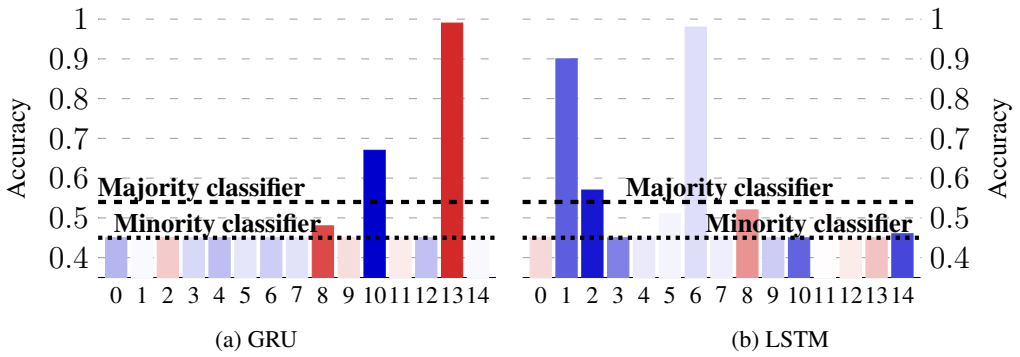


Figure 6: Diagnostic classifier weights to predict `minus_scope1+`.



(a) GRU (b) LSTM

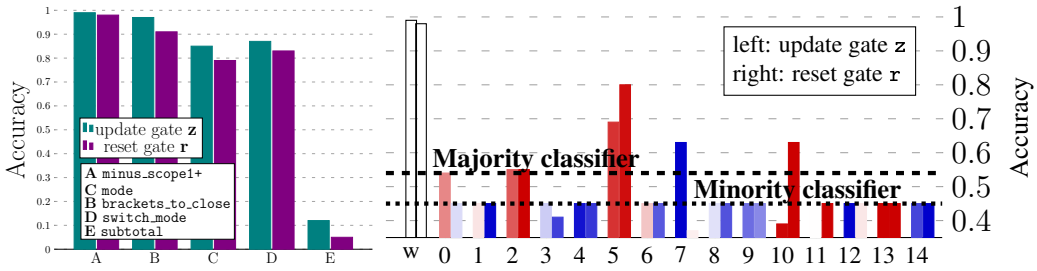
Figure 7: Accuracy of individual diagnostic classifier weights in predicting `minus_scope1+`.

3.4 Diagnosing gates

Hupkes et al. propose that diagnostic classifiers may also be used to diagnose gates. We investigate this possibility by focusing on the gates of the best performing GRU model. As the function of gates is fundamentally different from the function of the hidden units – they are meant to modulate information flow rather than to encode and memorise information and develop representations – the types of hypotheses for gates differ from hypotheses for the hidden layer activations. For instance, a gate would not likely maintain a prediction of the `subtotal`, as this is not impacting *how* information should be processed by the hidden layer,[3] but it may contain information about the operator `mode`. We train diagnostic classifiers to predict `minus_scope1+` and `close_minus_scope1+`, whether the mode should be switched in the current time step (`switch_mode`) and what the current `mode` is.

From the results (Figure 8a) we can see that `minus_scope1+` and `close_minus_scope1+` are much better represented by the gates than the operator `mode`. This is in itself not surprising, as under the hypothesised strategy the former two are sub-features used to compute the latter, and confirms our current strategy hypothesis. Interestingly, contrary to the hidden layer, the computation of `minus_scope1+` seems to be distributed over the gate values, with no single gate value predicting it with high accuracy on its own (Figure 8b). In the light of the difference in function of the gates and the hidden layer, this is not entirely surprising: where the hidden layer is required to keep track of `minus_scope1+`, the gate units have to use this information to decide how much every network unit should contribute to the next timestep. Furthermore, whereas a hidden layer unit can send information to all gates, a gate unit impacts only one hidden layer unit, making distribution of information over the gates more essential.

Another striking observation is that for this particular model, the update gate (left bars in Figure 8a) outperforms the reset gate (right bar) for all features. Both the division of work between gates as well as their exact function is a largely unexplored area, that is most certainly worth exploring. From these results only it is not possible to draw definite conclusions, but it demonstrates that diagnostic classifiers can provide further insights in such a quest.



(a) Diagnostic classification accuracy (b) Masked weights for `in_1minus_scope` diagnostic classifier

Figure 8: Diagnosing the gates of the best GRU model

[3] And in fact, a diagnostic classifier to predict this value from the gates has a very low accuracy, see Figure 8a.

4 Symbolic Guidance

In the previous section we explored different methods to qualitatively and quantitatively study recurrent neural networks, and suggested several possible paths for future research. In this section, we explore one such path, examining a method to employ the insight in the internal dynamics of a model to improve it in a more targeted way, that we call *symbolic guidance*.

The idea of symbolic guidance is to train a network not only directly on its target, but to provide additional intermediate targets that provide information to steer the model in the direction of a desired solution. Oftentimes, such targets will be motivated by an external symbolic model of the task at hand, but also different targets may be chosen.[4] In our case, the first choice of additional symbolic targets in this case study is the prediction of the `subtotal` of the cumulative strategy (see Figure 3). This choice of target is hoped to specifically help with processing right-branching sentences, that appear to be difficult for all models.

4.1 Training

We train 10 GRU and 10 LSTM model instances considering the error signal of two linear output units: one that is given the `subtotal` of the expression as described by the cumulative strategy, and one that has as target to predict the outcome of the entire expression at the end. To compute the gradients, we define a loss function that consists of 0.1 times the (mse) loss of the latter unit plus 0.9 times the (mse) loss of the prediction of the intermediate result. We train and test on the same data sets as before.

4.2 Results and error analysis

We plot the results of the training in Figure 9. For the LSTM we observe a clear improvement regarding both average and best performance. For the GRU model, providing additional targets did initially not seem to have had a positive effect. As our symbolical protocol was specifically designed to target sentences for which the previous models showed a low accuracy (in particular: right-branching sentences), we further study the results by adding a number of new subclasses to our test set.

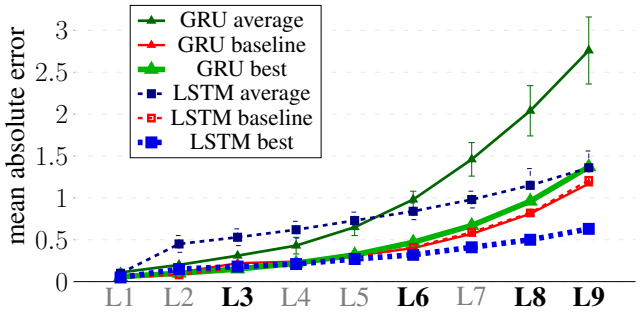


Figure 9: Training with symbolic guidance.

Processing deep structures Natural language models are quite commonly evaluated on a fairly superficial basis, by considering the overall performance on the entire test set. The symbolic nature of the arithmetic language allows us to study in more detail the effect of the symbolic guidance on as structural level. In Figure 10 we plot the performance of both the original models and the symbolically guided models on a wide range of different structures that may pose different processing difficulties. The LSTM models appear to have reached a solid improvement across all categories, showing a much more constant error profile for difficult and rarely seen categories. The GRU, on

[4]The method can be seen as an instance of multitask learning, which has been successfully applied to several different natural language processing tasks [e.g. Collobert and Weston, 2008, Søgaard and Goldberg, 2016], but shows mixed results in general, due to the intricacy of finding appropriate auxiliary tasks [Bingel and Søgaard, 2017, Alonso and Plank, 2016]. In symbolic guidance, this is not a concern, as auxiliary targets are motivated by diagnostic classification of already trained models and are thus tailor made for the task at hand.

the other hand, shows an improvement in some categories, while worsening in others. What exactly causes these differences is unclear from only these experiments, but it raises an important point of consideration: if LSTM and GRU models tend to have a preference for different types of solutions, or have a different susceptibility for symbolic guidance, this may have an impact on how they should be trained, but also on the types of (cognitive) hypotheses that we can infer from them. Further research is required to improve our understanding on this matter. The results also show that the evaluation of models should be treated with care, and that their overall accuracy on a large test set cannot always be taken as a indicative of their behaviour, as models with similar accuracies can be specialised on very different inputs.

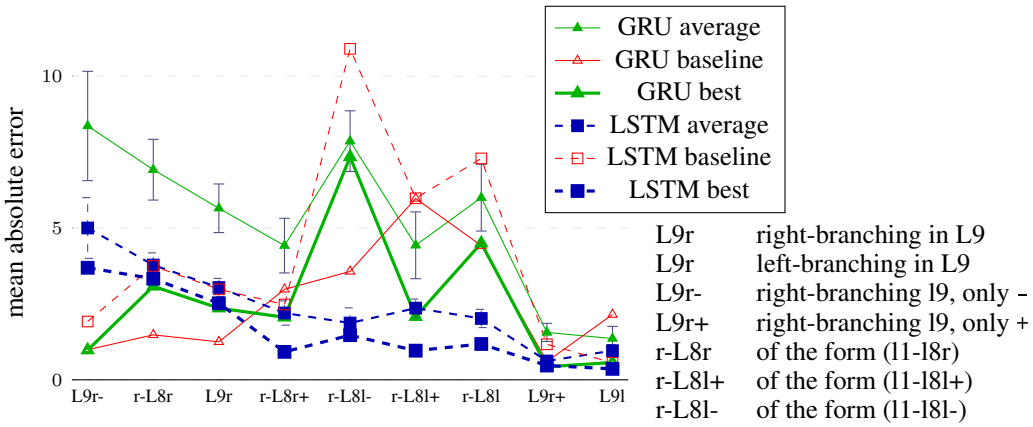


Figure 10: Error analysis

5 What now?

We presented an elaborate analysis of GRU and LSTM networks trained on the task of predicting the outcome of nested arithmetic expressions, which resulted in several interesting conclusions, but also raised many questions for further research. Using diagnostic classification, we revealed the strategy that both GRU and LSTM networks follow when processing such expressions: by keeping track of the operator `mode` and using this to cumulatively integrate every digit at the moment it is encountered with a preliminary prediction of the solution. By analysing the weights of the diagnostic classifiers, we found that they do so by keeping track of the scope of minuses. For both the best GRU and LSTM model instance, a single unit acts as a flag for *being in the scope of at least one minus*. Diagnosing also the gates, we demonstrated that the knowledge contained in the gates is more distributed, and concerns aspects of the strategy that are related to processing rather than memory. Lastly, we showed that knowledge about the internal dynamics can be leveraged by using symbolic guidance, in which symbolic targets are used to help networks to learn a strategy that they are already inclined to follow.

Throughout the paper, we provided several suggestions for further research, such as continuing with the analysis of the gates of GRU and LSTM models, comparing the susceptibility of GRUs and LSTMs to symbolic guidance, and using diagnostic classification to find appropriate additional targets to improve model performance. One future path that we consider particularly promising is using diagnostic classifiers to address catastrophic forgetting, a substantial problem of neural networks that we already briefly touched upon. Catastrophic forgetting refers to the incapability of neural networks to do compositional skill learning: Whenever a trained neural network is presented with a new task, it seems to inevitably overwrite any previously learned information or skills. A promising approach to solve this problem is to selectively allow learning on specific weights that are important for certain tasks [see e.g. Kirkpatrick et al., 2017, Lee et al., 2017], but this still requires knowledge on which weights map are specialised for which task. It is here exactly that we see a possibility for diagnostic classifiers, that can help to detect where and how specific functions are encoded.

While in this exploratory paper we found no space to address also the automation of the use of diagnostic classifiers, further research could focus on embedding them in meta-systems which automatically search for optimal neural network configurations.

References

Héctor Martínez Alonso and Barbara Plank. Multitask learning for semantic sequence prediction under varying data conditions. *CoRR*, abs/1612.02251, 2016. URL `http://arxiv.org/abs/1612.02251`.

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.

Joachim Bingel and Anders Søgaard. Identifying beneficial task relations for multi-task learning in deep neural networks. *CoRR*, abs/1702.08303, 2017. URL `http://arxiv.org/abs/1702.08303`.

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.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

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

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

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

Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. (under review), 2017. URL `http://dieuwkehupkes.nl/research/JAIR.pdf`.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835, 2017.

Sang-Woo Lee, Chung-Yeon Lee, Dong-Hyun Kwak, Jung-Woo Ha, Jeonghee Kim, and Byoung-Tak Zhang. Dual-memory neural networks for modeling cognitive activities of humans via wearable sensors. *Neural Networks*, 2017.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989.

Anders Søgaard and Yoav Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 231–235, 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.

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.