Learning compositionally through attentive guidance

Dieuwke Hupkes
University of Amsterdam
dieuwkehupkes@gmail.com

Anand Kumar Singh
University of Amsterdam
anand.singh@student.uva.nl

Kris Korrel
University of Amsterdam
kristian.korrel@student.uva.nl

German Kruszewski
Facebook AI Research
germank@fb.com

Elia Bruni
University of Amsterdam
elia.bruni@gmail.com

Abstract

In this paper, we introduce Attentive Guidance (AG), a new mechanism to direct a sequence to sequence model equipped with attention to find more compositional solutions that generalise even in cases where the training and testing distribution strongly diverge. We test AG on two tasks, devised precisely to asses the compositional capabilities of neural models and show how vanilla sequence to sequence models with attention overfit the training distribution, while the guided versions come up with compositional solutions that, in some cases, fit the training and testing distributions equally well. AG is a simple and intuitive method to provide a learning bias to a sequence to sequence model without the need of including extra components, that we believe allows to inject a component in the training process which is also present in human learning: guidance.

1 Introduction

In the past decade, neural network research has made astonishing progress on many tasks that require substantial generalisation skills, such as machine translation, visual reasoning and playing complex games such as Go [Sutskever et al., 2014, Silver et al., 2016, Johnson et al., 2017]. However, despite these empirical succcesses on human tasks, the form of generalisation such models exhibit diverges strongly from human type generalisation, which becomes visible when attacking systems with adversarial examples [Bender et al., 2017] or testing them on data with a different distribution than the training data [Lake and Baroni, 2017].

One key problem appears to be that neural networks trained with our current learning paradigms do not find solutions that exhibit systematic compositionality [Liška et al., 2018, Lake and Baroni, 2017], an elelement which has shown to be a crucial for systems to behave in a more human-like manner [Lake et al., 2015]. Neural networks seem to rely strongly on pattern matching and memorisation, but humans – on the other hand – are known to have a strong preference for compositional solutions [Schulz et al., 2016]. To quote an example given by Hudson and Manning [2018]: "suppose you have been asked the following: 'What is the yellow thing that is right of the small cube in front of the green object made of?'. How would you approach such a question?"

The lack of systematic compositionality in the solutions found by neural networks does not only result in data-hungry models that require large amounts of training data, but also severely limits their applicability in domains that require exactly this type of generalisation, such as language. Although there are some hybrid approaches to address this problem [e.g., Reed and De Freitas, 2015, Socher et al., 2010, Johnson et al., 2017], little is known about inducing compositional solutions in purely connectionist models, such as sequence to sequence (seq2seq) models. Understanding how to make compositional solutions emerge without losing the attractive properties of vanilla seq2seq models

Preprint. Work in progress.

would provide a tremendous advantage over hand-engineering compositional structures, as the latter does not easily generalise to different domains and requires very large amounts of data.

In this paper, we explore a novel approach to induce a learning bias in a vanilla seq2seq model with attention, that guides it to converge to more compositional solutions, rather than one of the many other potential fits for the training data. In this approach, for which we adopt the name *Attentive Guidance* (**AG**), we provide additional information to the attention mechanism of a regular encoder-decoder model [Sutskever et al., 2014, Cho et al., 2014], probing it to increase its understanding of what are sensible subsequences in the data. We test attentive guidance on two different datasets that were both specifically designed to test the compositional generalisation abilities of neural networks: the lookup table task [Liška et al., 2018] and a symbol rewriting task [Weber et al., 2018]. On both datasets, using AG consistently and robustly changes the types of solutions found to be more compositional.

In the next section, we describe our new approach, after which we discuss the datasets that are the subject of our studies. The results and analysis of our experiments can be found in Section 4 and 5, respectively. We end with discussing our work in a broader context and suggesting directions for future work.

2 Attentive guidance

It is well known that a finite set of input-output pairs can be described in many different ways [Angluin and Smith, 1983, Gold et al., 1967]. In the space of all potential solutions, one extreme is to simply memorise all pairs, which does not offer any means to extrapolate to new inputs; on the other end of the spectrum we find solutions that abide by the principle of minimal description length, which provides much stronger generalisation capacity [Hutter, 2004, Rissanen, 1978]. Vanilla seq2seq models can in theory represent many different solutions on this spectrum, but – as it turns out – are very unlikely to converge to the type of compositional solutions that we desire when they are trained with gradient descent. Given the fact that with our current learning paradigms we provide very little information regarding the types of solutions that we would like a model to find, this should not come as a complete surprise. An important research question is therefore: how can we direct neural models to find more systematic solutions?

We claim that AG can change the learning bias of a seq2seq model with attention to find more compositional solutions, without enhancing it with an additional components to do so. Using AG, we guide a model in an intuitive way to robustly change the type of solutions it converges to, by providing it additional information in the form of supervision on its attention component. In this section, we will first describe our method, followed by a description of the task we will use to demonstrate its effectiveness.

2.1 Intuition

Imagine teaching a child how to read. Rather than giving her increasingly long sequences of words and letters (the input) and starting to vocalise them (the target), you guide her to the process. You draw her attention by pointing at individual letters, tell her how to pronounce them, and show her how together they form a word. That is, you inform her about the individual components of the task of reading, and you tell her how these can be (compositionally) combined. Attentive guidance is based on exactly this principle: Via the attention component of a model, we explicitly tell the model how it should attend the input in a compositional way. From an intuitive perspective, this can be interpreted as guiding the model through generating outputs following an appropriate protocol, by telling it what the different parts of the input are and when they should be attended. Technically, guiding the attention vector of a model discourages solutions that strongly rely on memorisation of spurious patterns and pushes them to develop and use representations for sensible subsequences instead.

2.2 Implementation

We implement AG by extending a model's loss with an additional term, which captures the difference between the models' computed attention and the target attention pattern that is provided to the model at training time:

$$\mathcal{L}_{\text{AG}} = \mathcal{L} + \sum_{t=1}^{T} \sum_{i=1}^{N} - a_{i,t} \log \hat{a}_{i,t}$$

where T is the length of the output string, N is the length of the input string, $\hat{a}_{i,t}$ is the decoder-computed attention of input token i at time t and $a_{i,t}$ is its corresponding attention guidance target.[1] A schematic of attentive guidance is given in Figure 1.

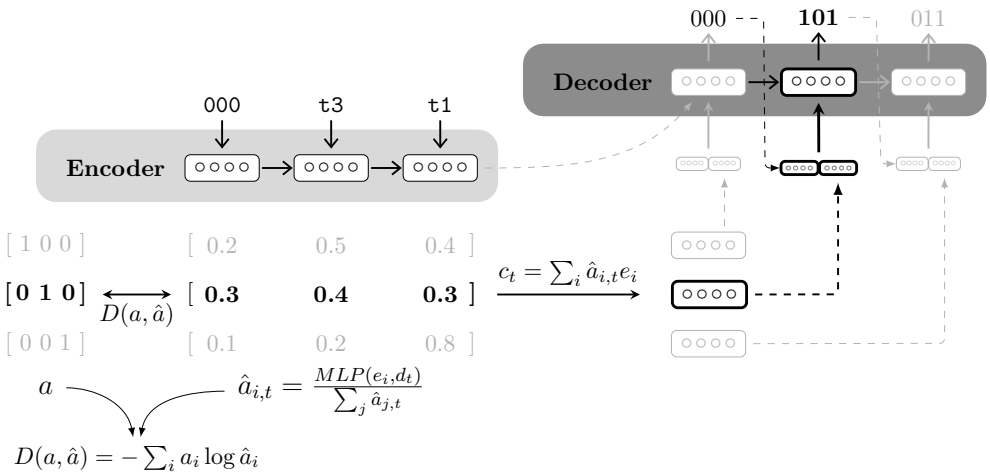


Figure 1: At each time step t, at AG-guided model infers a variable-length alignment weight vector $\hat{a}$ based on the current target state h_t and all source states. The Cross Entropy D between $\hat{a}$ and the one-hot AG target a is computed. A context vector c_t is then calculated as the weighted average, according to $\hat{a}$ over all the encoder states and is then concatenated with the output of the previous step to generate the next prediction.

3 Experiments

To test the effectiveness of AG, we look at two different datasets, both specifically designed to test the compositional generalisation abilities of neural networks: *lookup tables* [Liška et al., 2018] and a *symbol rewriting* task [Weber et al., 2018]. We discuss both datasets below.

3.1 Lookup Tables

Liška et al. [2018] propose the lookup table composition task as a simple setup to test compositional behaviour. The tasks consists in learning function composition by sequentially applying atomic lookup tables, where each such table is a bijective mapping from a binary string of length L to another binary string of the same length. These atomic tables are composed sequentially, i.e., each stage of the composition depends on the output of the atomic table that was computed in the previous step of the computation. For example, if $\texttt{t1}(000) = 010$ and $\texttt{t2}(010) = 001$, we can compute a function composition such as $\texttt{t1t2}(000) = 001$.

The simplicity of the atomic operations of this task allows to focus on compsitionality in isolation, because applying functions themselves is a rote memorisation task. Yet, Liška et al. [2018] show that only for very few intialisations vanilla seq2seq models converge to a solution that generalises to new inputs, while the vast majority of trained networks does not exhibit compositional behaviour when encountering previously seen compositions with a new input.

3.1.1 Data

Following Liška et al. [2018], for all our experiments we use 8 randomly generated 3-bit atomic lookup tables and all 64 possible length two compositions of these atomic tables. For each of these

[1] Code available at: `https://github.com/i-machine-think/machine`

Atomic	Atomic	Composed
$t1$	$t2$	$t1\ t2$
$000 \rightarrow 010$	$000 \rightarrow 110$	$000 \rightarrow 001$
$001 \rightarrow 110$	$001 \rightarrow 010$	$001 \rightarrow 010$
$010 \rightarrow 111$	$010 \rightarrow 001$	$010 \rightarrow 001$
$\dots$	$\dots$	$\dots$

(a)

$\mathcal{L}$: $X = \{A, B\}$,
$Y_A = \{a_1, a_2, a_3\}, Y_B = \{b_1, b_2, b_3\}$.
$a_1 \rightarrow a_{11}|a_{12},\ a_2 \rightarrow a_{21}|a_{22},\ a_3 \rightarrow a_{31}|a_{32}$
$b_1 \rightarrow b_{11}|b_{12},\ b_2 \rightarrow b_{21}|b_{22},\ b_3 \rightarrow b_{31}|b_{32}$

Input	**Valid output for $\mathcal{L}$**
AAB	$a_{21}a_{32}a_{12}a_{11}a_{22}a_{32}b_{13}b_{21}b_{32}$

(b)

Figure 2: (a) Examples of two 3bit lookup tables and a length two composition. (b) Example for symbol rewriting task.

64 compositions, we randomly take out 2 (out of 8) inputs for testing (*heldout inputs*). Additionally, we remove 8 randomly chosen compositions from the training set entirely (*heldout compositions*), as well as all compositions of which one of the atomic tables is `t7` or `t8` (*heldout tables*) and all compositions that only contain tables `t7` and `t8` (*new compositions*). This process leaves us with 4 different test sets of incremental difficulty, allowing us to test different generalisation conditions.

Following Liška et al. [2018], we define the training signal to be the entire sequence of outputs for the input sequence, including the intermediate computation steps. To the beginning of the target sequence, we append an additional step which reflects the *input* to the sequence of tables, to be able to attend to it during the generation of the output.

3.1.2 AG for Lookup Tables

We define the AG pattern as a sequential reading of the input: during the first step, the network should identify the input of the composition (`000` in Figure 1), in the second and third step, the targets corresponds to the first and second table that should be applied (`t3` and `t1` in Figure 1). Following this pattern, we point the network to the different elements of the input, which hopefully leads to more compositional solutions. An example for AG for lookup tables is depicted in Figure 1.

3.2 Symbol Rewriting

To demonstrate the effectiveness and generalisability of AG, we test its performance also on a second task, proposed recently by Weber et al. [2018] to test to what extent neural networks models can infer linguistic like structure from a dataset. The task they propose involves rewriting a sequence of input symbols $\{x_1, \dots, x_n\}$ to a sequence of output symbols $\{Y_{i,1}, \dots, Y_{i,n}\}$, following a predefined grammar. Every of the 40 input symbol x_i should be rewritten as a sequence of 3 (distinct) symbols from its own output alphabet Y_i, where each symbol in this alphabet can take on 16 different values.

Weber et al. [2018] show that seq2seq models trained on this task perform consistently very well on different examples of the same distribution, but generalise very poorly to sequence with fewer or more input symbols than seen in training. This indicates that also for this task, models rely more on memorising spurious patterns than inferring the underlying pattern, which makes this task very suitable as a testbed for AG.

3.2.1 Data

For our experiments we use the exact same data as Weber et al. [2018], whose training data consist of 100000 pairs with input lengths within $[5 - 10]$. Crucially, there are no repetitions of symbols in the input sequences. There are four different test sets, that test generalisation capacity in different scenarios. The examples in the *Standard* test set are drawn from the same distribution as the train data; *Repeat* is a set where the length distribution is kept identical, but repetitions introduced in the input sequences are allowed; *Short* consists of shorter input sequences ($[1, 4]$), and *Long* consists of longer sequences ($[11, 15]$). All data sets are non-exhaustive and sampled randomly from all valid input-output pairs. For model selection, a validation set is created by sampling inputs with length $[3 - 12]$, thus containing examples from all different testing conditions.

3.2.2 AG for Symbol Rewriting

Given the strongly structured nature of the symbol rewriting task, combined with the fact that each output symbol is generated by exactly one input symbol, an AG pattern can be very straightforwardly defined. For every output symbol, we define the attention target to be the position of the input word from which it was generated. That is, for the first three output symbols, the attention target corresponds to the first input symbol, the second three output symbols should attend to the second input symbol, and so on.

4 Results

We test AG by comparing models trained with it (*guided*) with a baseline model (*baseline*) – a standard encoder-decoder model with an attention mechanism that is learned end-to-end along with the rest of the parameters. The guided model is identical from an architectural perspective, but has the additional training objective to minimise the cross-entropy loss between the calculated attention vectors and a provided target attention vectors in its backward pass. To take apart the learnability of the target attention patterns and their effect on the learned solution, in the analysis section of this paper we evaluate also the impact of replacing the calculated attention vectors in the forward pass of a model with the target attention vectors (*hard guidance*).

4.1 Lookup tables

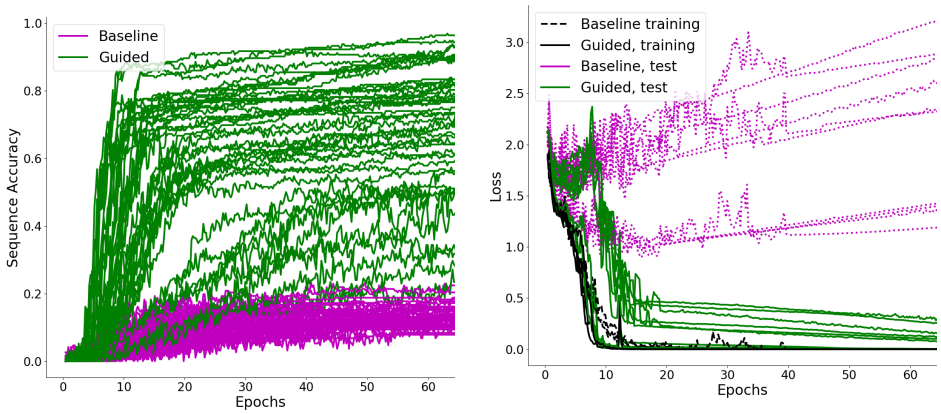
We run a grid search over multiple model sizes and attention mechanisms, to understand the robustness of AG over different parameter settings. In an initial exploration, we test both long short-term memory networks [LSTM, Hochreiter and Schmidhuber, 1997] and gated recurrent units [GRU, Chung et al., 2014]. As we find that the latter behaves better for this particular task for both the baseline and guided models, we run our grid search only for GRU models.

For each condition (*baseline* and *guided*), we search through embedding and hidden sizes {16, 32, 64 and 128} and {32, 64, 128, 256 and 512}, respectively. Furthermore, we test two different alignment models for the attention mechanism: *dot* where the attention weights are computed by taking a dot-product between the encoder and decoder hidden state, and *mlp* where instead an MLP is used. We experiment with putting the attention before (*pre-rnn* attention) and after (*post-rnn* attention) the recurrency in the decoder (following Bahdanau et al. [2015] and Luong et al. [2015], respectively). More details about the different attention mechanisms can be found in the supplementary materials. We run each configuration 3 times and investigate the development of the accuracy of the resulting models on *heldout inputs*, *heldout tables*, *heldout compositions* and *new compositions*. Below we report on the results.

4.1.1 Model Size

Our grid search confirms the findings of Liška et al. [2018] that vanilla seq2seq models cannot solve the lookup table task, which is irrespective of alignment model, place of the attention and network size. The baseline performance slightly increases with hidden layer size, but never reaches an accuracy of higher than 30% accross the heldout data, even though all models have a near-perfect performance on the training data. The guided models, on the other hand, appear to require a certain hidden layer size to generalise well, although even the smallest guided models outperform the larger baseline models. To illustrate, we depict the development of the accuracy for the *heldout tables* case for all baseline (green) and guided (purple) models in Figure 3a.

Furthermore, all baseline models exhibit overfitting on the heldout data, whereas the guided models, do not overfit at all: although smaller models do not always reach a high validation accuracy, their validation loss never increases during training. In Figure 3b, we plot the development of the loss for all validation sets for the best baseline and guided configurations. Importantly, while the training loss converges quickly, the validation loss keeps decreasing steadily for much longer, which can be explained by the influence of the attention loss. We will revisit this topic in Section 5.



(a) (b)

Figure 3: (a) Accuracy on heldout tables for all models in the grid search. Green lines represent the baseline, purple the guided models. (b) Loss development for chosen configurations on all different heldout datasets.

4.1.2 Attention Mechanisms

The baseline models are indifferent with respect to both the alignment model and the location of the attention. For the guided models, the *mlp* alignment model combined with pre-rnn attention outperform all other conditions.

4.1.3 Evaluation of 'Best' Models

For both the baseline and the guided models we pick the configuration that has the best performance across the heldout data. We find the best baseline configuration to be a model with an embedding size of 128 and a hidden layer size of 512, whereas the best guided configuration has 16 and 512 nodes for embeddings and hidden layers, respectively. To evaluate the chosen configurations, we generate 4 new sets of training and testing data, generated with similar criteria, but with different instances. For each configuration and sample we train two models, using Adam as optimiser (lr=0.001), resulting in 8 different runs for every configuration. We perform early stopping based on the performance on the *heldout tables* dataset, which we believe best reflects the lifelong learning condition in which we would like a network to generalise.[2] The average accuracies can be found in Figure 4.

As expected, the baseline model fails to capture any compositional strategy and performs very poorly across all test sets.[3] All guided models, however, achieve a perfect generalisation accuarcy to the simplest case of generalisation (which is in fact the only condition studied by Liška et al. [2018]) and also generalise well to compositions that are not seen at all during training (*heldout compositions*). For the two most difficult cases of compositionally that require using tables compositionally that are only seen atomically (*heldout tables* and *new compositions*), the generalisation accuracy goes slightly down, and not all models converge to a perfect performance.

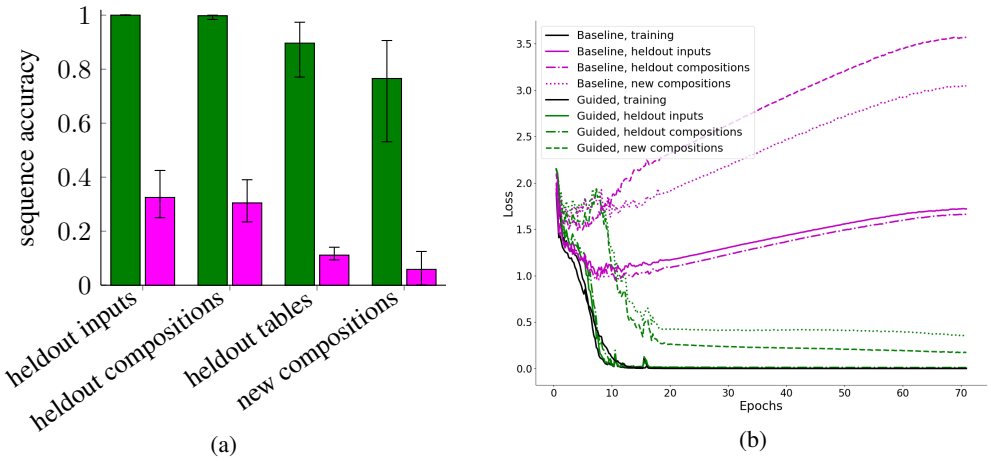
Overall, these results provide a clear demonstration that attentive guidance very effectively direct models to more compositional solutions. In Section 5, we try to push the generalisation skills of the model even further, and provide a more in depth analysis of the guided models.

4.2 Symbol rewriting

We perform again a modest grid search, over embedding sizes {32, 64} and hidden layer sizes {64, 128, 256}. For the baseline model, we find that the configuration of 64x64 performs best,

[2] In fact, since the guided models do not exhibit overfitting on validation, we do not need to use the validation set for the guided models.

[3] It should be observed, however, that the baseline models still perform well above chance level, which lies at 0.2% for this task.

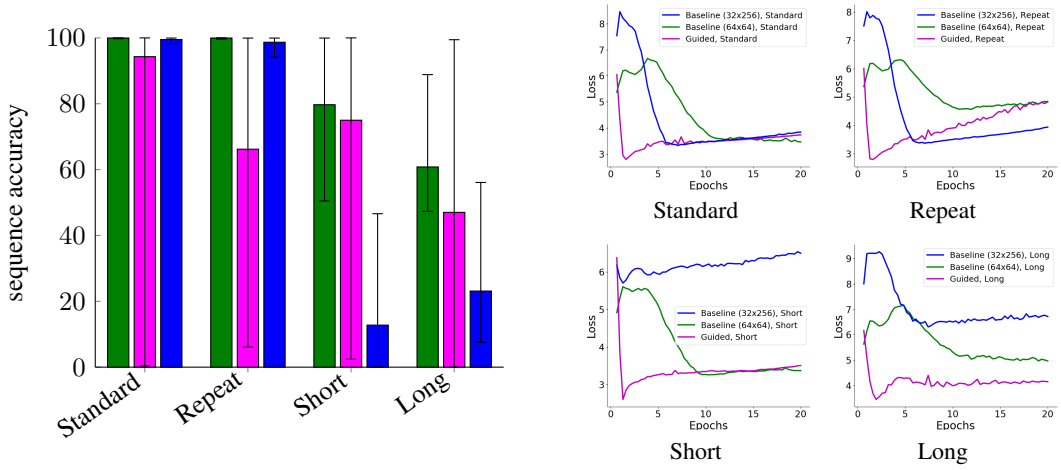


(a) (b)

Figure 4: (a) Average accuracies on all different test sets for best configurations. Error bars indicate max and min values of the different models. (b) Loss development for different test sets for best configurations.

whereas the guided models require more parameters (32x256). Given the high variance in the baseline performance, we decide to include also a baseline model that has the same dimensions as the best guided models in our further experiments. Following Weber et al. [2018], we train all configurations 50 times, and use the validation performance as stopping criterion.

In Figure 5a, we show the average performance of the guided models (green) and the two baselines (magenta and blue), using the error bars to indicate the minimum and maximum values encountered during training. Although our scores are somewhat higher than those found by Weber et al. [2018], they do confirm that no baseline model can find a solution covering all test set distributions. Overall, the guided models exhibit both a considerably larger accuracy accross test sets, and show significantly less variance among different runs. Additionally, the guided models converge much quicker than the baseline models (see Figure 5). In conclusion, using AG leads to a strong improvement in the symbol writing task, but does not completely solve it.



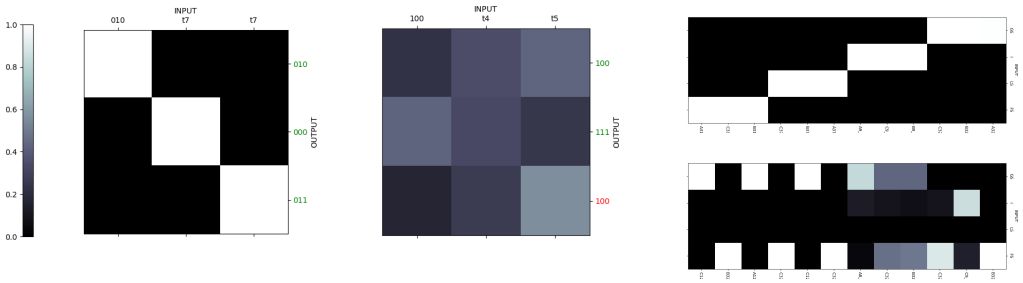
(a) (b)

Figure 5: (a) Accuracy on all different test condition for chosen configurations. Green bars represent the guided model. (b) Loss development for chosen configurations on all different heldout datasets.

5 Analysis

In the previous section, we showed that AG very consistently improves the generalisation skills of seq2seq models. In particular, we observed that AG helps to decrease the variance between different initialisations and strongly countereffects overfitting. In this section, we compare the attention patterns learned via AG with the patterns of our baselines. Ad:itionally, we study the effect of applying *hard guidance*, where instead of learning the AG targets, we directly replace the attention vector of a model with its target vector in the forward pass.

5.1 Attention Patterns



(a) (b) (c)

Figure 6: Attention plots for (a) a new composition processed by a guided model (b) a heldout input processed by a baseline model (c) the attention patterns for a short composition in the symbol rewriting task.

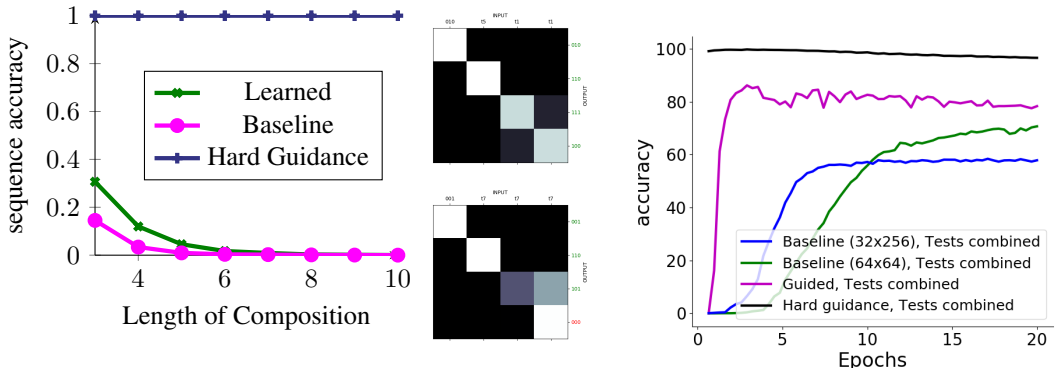
The attention plots demonstrate that the guided models learned to generate (almost) perfect attention pattern. The baseline models, on the other hand, do not use their attention mechanism in a targetted way, as indicated by their diffused attention. Interestingly, when probing the guided lookup table models to generalise to longer sequences, we observe that also their attention becomes more diffuse (see Figure 7). This failure in producing the right attention patterns matches also the performance of the models on longer compositions, that rapidly drops when the compositions get longer (see Figure 7). To test to what extent this inability stems from the model's difficulty to generalise the attention patterns to longer sequences, we investigate if we can push models to find solutions for the task when instead of learning the attention patterns, we use *hard guidance*.

5.2 Hard Guidance

Our experiments with hard guidance show that when the right attention vector is provided to a model, for both the lookup table and symbol rewriting task it almost always converges to perfect performance. The difference between train and test performance are flattened even in very extreme conditions, such as the very long lookup table sequences. We conjecture that this finding begs into question whether the current architecture optimally facilitates AG, and we propose that perhaps it would be more suitable to use two distinct models to learn the target outputs and the correct attention patterns.

6 Discussion

In this paper, we introduce AG, a new mechanism to direct a seq2seq model equipped with attention to find more compositional solutions that generalise even in cases where the training and testing distribution strongly diverge. We test AG on two tasks devised precisely to asses the compositional capabilities of neural models and show how vanilla seq2seq models with attention overfit the training distribution, while the guided versions come up with compositional solutions that, in some cases, fit both the training and testing distributions equally well. AG is a simple and intuitive method to provide a learning bias to a sequence to sequence model without the need of including extra components, that we believe allows to inject a component in the training process which is also present in human learning: guidance.



(a) (b) (c)

Figure 7: (a) Accuracies on longer compositions in lookup tables (b) Attention patterns for longer compositions. (c) Accuracies on symbol rewriting for baseline, learned and hard AG.

While AG stands out as a promising direction to help sequence to sequence models to learn compositionally, we also find some limitations. In particular, when the target attentions are learned together with the objective of the main task, it seems difficult to prioritise the two different losses. The almost perfect results using a hard version of AG are a clear indication that in future work we should focus on designing a new architecture that allows to take apart the two different learning objectives.

Acknowledgements

We thank Marco Baroni for inspiration and his valuable feedback, and Facebook AI Research for sponsoring this project.

References

Dana Angluin and Carl H Smith. Inductive inference: Theory and methods. *ACM Computing Surveys (CSUR)*, 15(3):237–269, 1983.

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

Emily Bender, Hal Daumé III, Allyson Ettinger, and Sudha Rao. Proceedings of the first workshop on building linguistically generalizable nlp systems. In *Proceedings of the First Workshop on Building Linguistically Generalizable NLP Systems*, 2017.

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

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

E Mark Gold et al. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

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

Drew A. Hudson and Christopher D. Manning. Compositional attention networks for machine reasoning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 1988–1997. IEEE, 2017.

Brenden M. Lake and Marco Baroni. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. *CoRR*, abs/1711.00350, 2017.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Adam Liška, Germán Kruszewski, and Marco Baroni. Memorize or generalize? searching for a compositional rnn in a haystack. *arXiv preprint arXiv:1802.06467*, 2018.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D/D15/D15-1166`.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

J Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–658, 1978.

Eric Schulz, Josh Tenenbaum, David K Duvenaud, Maarten Speekenbrink, and Samuel J Gershman. Probing the compositionality of intuitive functions. In *Advances in neural information processing systems*, pages 3729–3737, 2016.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

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.

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

Noah Weber, Leena Shekhar, and Niranjan Balasubramanian. The fine line between linguistic generalization and failure in seq2seq-attention models. In *Workshop on new forms of generalization in deep learning and natural language processing, NAACL'18*, 2018.

A Attention mechanisms

A.1 MLP attention method

The alignment model for the attention mechanism we use for all our final models is an multilayer perceptron that takes as input a decoder state (eo_i) and an encoder state (eo_t), and calculates the score as:

$$score(eo_i, do_t) = W_s * ReLU(W_c * [eo_i; do_t]),$$

where W_s is a $1 \times H$, and W_c a $H \times 2H$ matrix. H is the hidden layer size of the RNN. We calculate these scores for every encoder state, and then use a softmax layer to transform the series of score to a probability distribution. The context vector is then computed as a weighted sum over the encoder states, taking the marginalised attention scores as weights:

$$c_t(eo, do_t) = \sum_i eo_i * \frac{e^{score(eo_i, do_t)}}{\sum_j e^{score(eo_j, do_t)}}$$

A.2 Attention mechanisms

We use two different attention mechanism that differ in where the calculated context vectors are used in the decoder.

The first mechanism, which we call **post-rnn**, computes the attention scores based on the current decoder state do_t. The context vector is then concatenated with do_t and fed through the output layer of the network, that produces a probablity distribution over the output words:

$$out = Soft - Max(W_o * [do_t; c_t(eo, do_t])$$

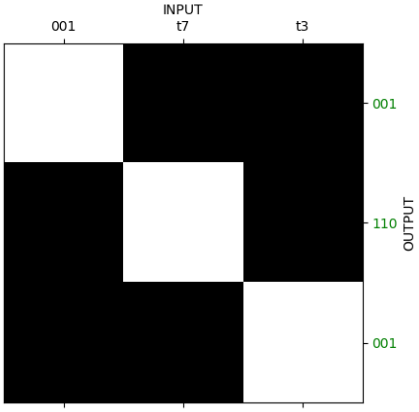
Instead, the **pre-rnn** attention scores are computed based on the *previous* decoder state do_{t-1}, instead of the current. The resulting context vector is then concatenated with the embedded input of the current decoder step (which is usually the output of the previous decoder step), to produce the decoder input di_t. Note that this makes the input size of the decoder twice as large.

$$di_t = [de_t; c_t(eo, do_{t-1})]$$

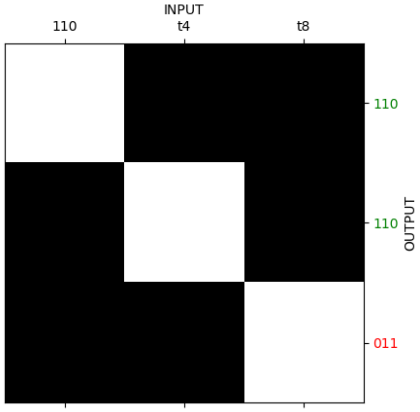
The pre-rnn mechanism allows the decoder to incorporate the context information in its recurrency, rather than only using it when generating the output after it.

B Attention patterns

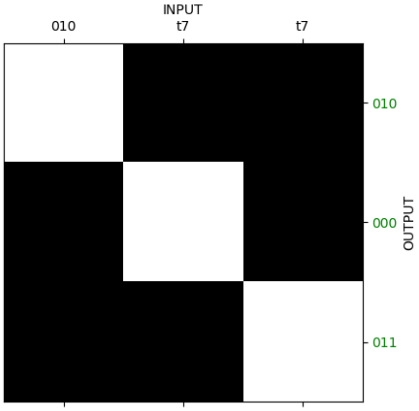
Below we provide here more examples of attention plots for models trained with AG. Correct outputs are coloured green, whereas incorrect outputs are coloured red.



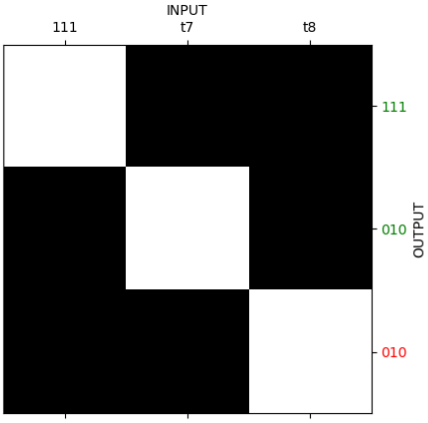
(a) Pre-rnn: Heldout Tables



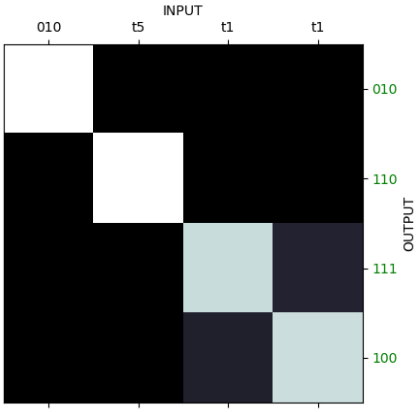
(b) Pre-rnn: Heldout Tables



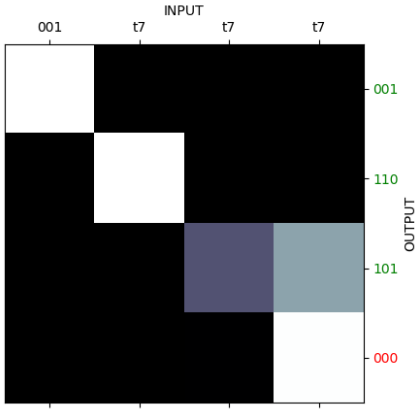
(c) Pre-rnn: New Compositions



(d) Pre-rnn: New Compositions



(e) Pre-rnn: Composition of length 3



(f) Pre-rnn: Composition of length 3