SYSTEMATIC GENERALIZATION: WHAT IS REQUIRED AND CAN IT BE LEARNED?

Anonymous authors
Paper under double-blind review

ABSTRACT

Numerous models for grounded language understanding have been recently proposed, including (i) generic modules that can be used easily adapted to any given task with little adaptation and (ii) intuitively appealing modular models that require background knowledge to be instantiated. We compare generic and modular models in how much they lend themselves to a particular form of systematic generalization. Using a synthetic VQA test, we evaluate which models are capable of reasoning about all possible object pairs after training on only a small subset of them. Our findings show that the generalization of modular models is much more systematic and that it is highly sensitive to the module layout, i.e. to how exactly the modules are connected. We furthermore investigate if modular models that generalize well could be made more end-to-end by learning their layout and parametrization. We show how end-to-end methods from prior work often learn a wrong layout and a spurious parametrization that do not facilitate systematic generalization. Our results suggest that, in addition to modularity, systematic generalization in language understanding may require explicit regularizers or priors.

1 INTRODUCTION

In recent years, neural network based models have become the workhorse of natural language understanding and generation. They empower industrial systems in machine translation (Wu et al., 2016) and text generation (Kannan et al., 2016) and show state-of-the-art performance on numerous benchmarks, including Recognizing Textual Entailment (RTE) (Gong et al., 2017), Visual Question Answering (VQA) (Jiang et al., 2018), and Reading Comprehension (Wang et al., 2018). Despite these successes, a growing body of literature suggests that these approaches do not generalize outside of the specific distributions they are trained on, something that is necessary for a language understanding system to be widely deployed in the real world. Investigations on the three aforementioned tasks have shown that neural models easily latch onto statistical regularities which are omnipresent in existing datasets (Agrawal et al., 2016; Gururangan et al., 2018; Jia & Liang, 2017) and extremely hard to avoid in large scale data collection. Having learned such dataset-specific solutions, neural networks fail to deal with examples that are even slightly out of domain, yet are trivial for humans. These findings have been corroborated by a recent investigation on a synthetic instruction-following task (Lake & Baroni, 2018), in which seq2seq models (Sutskever et al., 2014; Bahdanau et al., 2015) have shown little systematicity (Fodor & Pylyshyn, 1988) in how they generalize, i.e. they would not learn a general rule on how to compose words and fail spectacularly when asked to interpret "jump twice" after training on "jump", "run twice" and "walk twice".

An appealing direction to improve the generalization capabilities of neural models is to add modularity and structure to their design to make them structurally resemble the kind of rules they are supposed to learn (Andreas et al., 2016; Gaunt et al., 2017). For example, in the Neural Module Network paradigm (NMN, Andreas et al. (2016)), a neural network is assembled from several *neural modules*, where each module is meant to perform a particular subtask of the input processing, much like a computer program composed of functions. The NMN approach is intuitively appealing but its widespread adoption has been hindered by the large amount of domain knowledge it requires to decide (Andreas et al., 2016) or predict (Johnson et al., 2017; Hu et al., 2017) how the modules should be created (parametrization) and how they should be connected (layout) based on a natural language utterance. Besides, their performance has often been matched by more traditional neural models, such as FiLM (Perez et al., 2017), Relations Networks (Santoro et al., 2017), and MAC

networks (Hudson & Manning, 2018). Lastly, generalization properties of NMNs, to the best of our knowledge, have not been rigorously studied prior of this work.

In this work we investigate the impact of explicit modularity and structure on systematic generalization by studying the generalization of NMNs and contrasting it to those of generic models. For this case study we focus on the task of visual question answering (VQA), in particular its simplest binary form when the answer is either "yes" or "no". Such a binary VQA task that we focus on here can be seen as a fundamental task of language understanding, as it requires one to evaluate the truth value of the utterance in a context. There are many ways to formalize the systematic generalization requirements that are desirable for a VQA model, so we choose to focus on the following basic requirement: *a good model should be able to reason about all possible object combinations despite being trained on a very small subset of them*. We believe that this is a key prerequisite to using VQA models in the real world, because they should be robust at handling unlikely combinations of objects. We instantiate our generalization demands in the form of a new synthetic dataset called **S**patial **Q**ueries **O**n **O**bject **P**airs (SQOOP), in which a model has to perform basic spatial relational reasoning about pairs of randomly scattered letters and digits in the image (e.g. answering the question "*Is there a letter A left of a letter B?*"). The main challenge in SQOOP is that models are evaluated on all possible object pairs, but trained on only a subset of them.

Our first finding is that NMNs do generalize better than other neural models when an appropriate choice of layout and parametrization is made. We then investigate which factors contribute to improved generalization performance and find that using a tree layout, as opposed to a chain layout, is crucial for solving the hardest version of our dataset. Lastly, and perhaps most importantly, we experiment with existing methods for making NMNs more end-to-end by inducing the module layout (Johnson et al., 2017) or learning module parametrization through soft-attention over the question (Hu et al., 2017). Our experiments show that such end-to-end approaches often fail by not converging to tree layouts or by learning a blurred parameterization for modules, which results in poor generalization on the hardest version of our dataset. We believe that our findings challenge the intuition of researchers in the field and provide a foundation for improving systematic generalization of neural approaches to language understanding.

2 THE SQOOP DATASET FOR TESTING SYSTEMATIC GENERALIZATION

We perform all experiments of this study on the SQOOP dataset. SQOOP is a minimalistic VQA task that is designed to test one particular type of generalization: the ability to disentangle the meaning of relation words and object words and then compose these meanings in novel contexts to perform basic relational reasoning in a consistent way. Clearly, given known objects X, Y and a known relation R, a human can easily verify whether or not the objects X and Y are in relation R. Some instances of such queries are common in daily life (*is there a cup on the table*), some are extremely rare (*is there a violin under the car*), and some are unlikely but have similar, more common counter-parts that are likely (*is there grass on the frisbee* vs *is there a frisbee on the grass*). Still, a person can easily answer these questions by understanding them as just the composition of the three separate concepts. Such compositional reasoning is extremely desirable for our models, and we have created SQOOP to test for it.

Concretely speaking, SQOOP requires observing a 64×64 RGB image x and answering a yes-no question $q = \mathrm{X\,R\,Y}$ about whether objects X and Y are in a spatial relation R. The questions are represented in a redundancy-free X R Y form; we did not aim to make the questions look like natural language. Each image contains 5 randomly chosen and randomly positioned objects. There are 36 objects: the latin letters A-Z and digits 0-9, and there are 4 relations: LEFT_OF, RIGHT_OF, ABOVE, and BELOW. This results in $36 \cdot 35 \cdot 4 = 5040$ possible unique questions (we do not allow questions about identical objects). To make negative examples challenging, we ensure that both X and Y of a question are always present in the associated image and that there are always distractor objects $\mathrm{Y}' \neq \mathrm{Y}$ and $\mathrm{X}' \neq \mathrm{X}$ such that $\mathrm{X\,R\,Y}'$ and $\mathrm{X}'\mathrm{\,R\,Y}$ are both true for the image. These extra precautions guarantee that answering a question requires the model to locate all possible X and Y then check if any pair of them are in the relation R. Two SQOOP examples are shown in Figure 2.

Our goal is to discover which models can correctly answer questions about all $36 \cdot 35$ possible object pairs in SQOOP after having been trained on only a subset. For this purpose we build training sets containing $36 \cdot 4 \cdot k$ unique questions by sampling k different right-hand-side (RHS) objects Y_1,

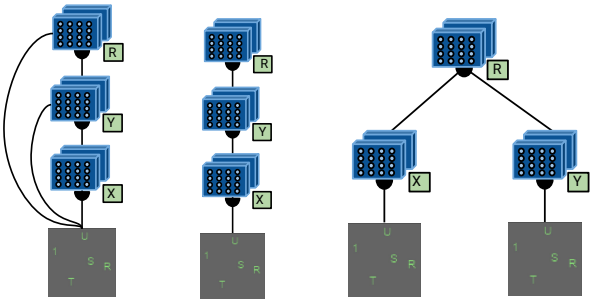
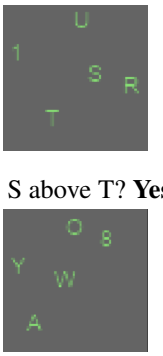


Figure 1: Different NMN layouts for the question "is there an S above T'. **left** represents *NMN-Chain-Shortcut*, **center** represents *NMN-Chain* and **right** represents *NMN-Tree*. *Refer to Section-3.2 for details.*



a: S above T? **Yes**

b: W left of A? **No**

Figure 2: We show a positive (left) and negative (right) example of the SQOOP dataset.

Y_2, ..., Y_k for each left-hand-side (LHS) object X. We use this procedure instead of just uniformly sampling object pairs in order to ensure that each object appears in at least one training question, thereby keeping the all versions of the dataset solvable. We will refer to k as the *#rhs/lhs parameter* of the dataset. Our test set is composed from the remaining $36 \cdot 4 \cdot (35 - k)$ questions. We generate training and test sets for rhs/lhs values of 1,2,4,8 and 18, as well as a dataset in which both the training and the test set contain all the questions (with different images). We will refer to this control version of SQOOP as #rhs/lhs=35. Note that lower #rhs/lhs versions are harder for generalization due to the presence of spurious correlations between lhs and rhs objects that models may adapt to - in the extreme case of #rhs/lhs=1, a model may learn to predict the rhs object from the lhs.

In order to exclude a possible compounding factor of overfitting on the training images, all our training sets contain 1 million examples, so for a dataset with k rhs/lhs we generate approximately $\frac{10^6}{36 \cdot 4 \cdot k}$ different images per question.

3 Models

A great variety of VQA models have been recently proposed in the literature, among which we can distinguish two trends. Some of the recently proposed models, such as FiLM (Perez et al., 2017) and Relation Networks (RelNet, Santoro et al. (2017)) are highly *generic* and do not require any task-specific knowledge to be applied on a new dataset. On the opposite end of the spectrum are *modular* and *structured* models, typically flavours of Neural Module Networks (Andreas et al., 2016), that require some knowledge about the task at hand to be instantiated. We evaluate several state-of-the-art models in both families. In all models, the image x is first fed through a CNN based network, that we refer to as the *stem*, to produce a feature-level 3D tensor h_{x}. This is passed through a model-specific computation conditioned on the question q, to produce a joint representation $h_{q\,\mathrm{x}}$. Lastly, this representation is fed into a fully-connected *classifier* network to produce logits for prediction. Therefore, the main difference between the models we consider is how the computation $h_{q\,\mathrm{x}} = model(h_{\mathrm{x}}, q)$ is performed.

3.1 Generic Models

We consider four generic models in this paper: CNN+LSTM, FiLM, Relation Network (RelNet), and Memory-Attention-Control (MAC) network. For CNN+LSTM, FiLM, and RelNet models, the question q is first encoded into a fixed-size representation h_q using a unidirectional LSTM network. **CNN+LSTM** flattens the 3D tensor h_{x} to a vector and concatenates it with h_q to produce $h_{q\,\mathrm{x}}$.

$$h_{q\,\mathrm{x}} = [vec(h_{\mathrm{x}}); h_q] \tag{1}$$

RelNet uses a network g which is applied to all pairs of feature columns of h_{x} concatenated with the question representation h_q, all of which is then pooled to obtain $h_{q\,\mathrm{x}}$:

$$h_{q\,\mathrm{x}} = \sum_{i,j} g(h_{\mathrm{x}}(i), h_{\mathrm{x}}(j), h_q) \tag{2}$$

where $h_x(i)$ is the i-th feature column of h_x. **FiLM** networks use N convolutional FiLM blocks applied to h_{x}. A FiLM block is a residual block (He et al., 2016) in which a feature-wise affine transformation (FiLM layer) is inserted after the 2nd convolutional layer. The FiLM layer is conditioned on the question at hand via prediction of the scaling and shifting parameters γ_n and β_n:

$$[\gamma_n; \beta_n] = W_q^n h_q + b_q^n \tag{3}$$

$$\tilde{h}_{q\,\mathrm{x}}^n = BN(W_2^n * ReLU(W_1^n * h_{q\,\mathrm{x}}^{n-1} + b_n)) \tag{4}$$

$$h_{q\,\mathrm{x}}^n = h_{q\,\mathrm{x}}^{n-1} + ReLU(\gamma_n \odot \tilde{h}_{q\,\mathrm{x}}^n \oplus \beta_n) \tag{5}$$

where BN stands for batch normalization, $*$ stands for convolution and $\odot$ stands for element-wise multiplications. $h_{q\,\mathrm{x}}^n$ is the output of the n-th FiLM block and $h_{q\,\mathrm{x}}^0 = h_{\mathrm{x}}$. The output of the last FiLM block $h_{q\,\mathrm{x}}^N$ undergoes an extra 1×1 convolution and max-pooling to produce $h_{q\,\mathrm{x}}$. **MAC** network of Hudson & Manning (2018) produces $h_{q\,\mathrm{x}}$ by repeatedly applying a Memory-Attention-Composition (MAC) cell that is conditioned on the question through an attention mechanism. The MAC model is quite complex and we refer the reader to the original paper for details.

3.2 Neural Module Networks

Neural Module Networks (Andreas et al., 2016) are an elegant solution that constructs a question-specific network by composing together trainable neural modules, drawing inspiration from symbolic approaches to question answering (Malinowski & Fritz, 2014) . To answer a question with an NMN, one first constructs the computation graph by making the following decisions: (a) how many modules and of which types will be used, (b) how the modules will be connected to each other, and (c) how these modules are parametrized based on the question. We refer to the aspects (a) and (b) of the computation graph as the *layout* and the aspect (c) as the *parametrization*. In the original NMN and in many follow-up works, different module types are used to perform very different computations, e.g. the `Find` module from Hu et al. (2017) performs trainable convolutions on the input attention map, whereas the `And` module from the same paper computes an element-wise maximum for two input attention maps. In this work we follow the trend of using more homogeneous modules started by Johnson et al. (2017), who use only two types of modules: unary and binary, both performing similar computations. We go one step further and retain a single binary module type, using a zero tensor for the second input when only one input is available. For simplicity, we also choose to use exactly three modules and therefore simplify defining the layout of the NMNs to just defining how the modules are connected. Our preliminary experiments have shown that, even after this simplification, NMNs are far ahead of other models in terms of generalization.

In the original NMN, the layout and parametrization were set in an ad-hoc manner for each question by analyzing a dependency parse. In the follow-up works (Johnson et al., 2017; Hu et al., 2017), these aspects of the computation are predicted by learnable mechanisms with the goal of reducing the amount of background knowledge required to apply the NMN approach to a new task. We experiment with the End-to-end NMN (N2NMN) (Hu et al., 2017) paradigm from this family, which predicts the layout with a seq2seq model (Sutskever et al., 2014) and computes the parametrization of the modules using a soft attention mechanism. Since all the questions in SQOOP have the same structure, we can get away with not employing any seq2seq models and have a trainable layout variable and trainable attention variables for each module.

Formally, our NMN is constructed by repeatedly applying a *generic neural module* $f(\theta, \gamma, h_l, h_r)$, which takes as inputs the shared parameters θ, the question-specific parametrization γ and the left-hand side and right-hand side inputs h_l and h_r. M such modules are connected and conditioned on

a question $q = (q_1, q_2, q_3)$ as follows:

$$\gamma_k = \sum_{i=1}^{s} \alpha^{k,i} e(q_i) \tag{6}$$

$$h_k = f(\theta, \gamma_k, \sum_{j=-1}^{k-1} \tau_0^{k,j} h_j, \sum_{j=-1}^{k-1} \tau_1^{k,j} h_j) \tag{7}$$

$$h_{qx} = h_M \tag{8}$$

In the equations above, $h_{-1} = 0$ is the zero tensor input, $h_0 = h_x$ are the image features outputted by the stem, and e is the embedding table for the questions words, and we refer to $A = (\alpha^{k,i})$ and $T = (\tau_0^{k,i}, \tau_1^{k,i})$ as the *parametrization attention matrix* and the *layout tensor* respectively.

We perform our experiments with the Find module from Hu et al. (2017) and the $\mathrm{Residual}$ module from Johnson et al. (2017) with very minor modifications - we use 64 dimensional CNNs in our $\mathrm{Residual}$ blocks since our dataset consists of 64 $\times$ 64 images. The equations for the $\mathrm{Residual}$ module are as follows:

$$\theta = \emptyset, \tag{9}$$

$$\gamma = [W_1; b_1; W_2; b_2; W_3; b_3], \tag{10}$$

$$\tilde{h} = ReLU(W_3 * [h_l; h_r] + b_3), \tag{11}$$

$$f_{Residual}(\gamma, h_l, h_r) = ReLU(\tilde{h} + W_1 * ReLU(W_2 * \tilde{h} + b_2)) + b_1), \tag{12}$$

and for Find module as follows:

$$\theta = [W_1; b_1; W_2; b_2], \tag{13}$$

$$f_{Find}(\gamma, h_l, h_r) = ReLU(W_1 * \gamma \odot ReLU(W_2 * [h_l; h_r] + b_2) + b_1). \tag{14}$$

In formulas above W_1, W_2, W_3 are convolution weights, and b_1, b_2, b_3 are biases. The main difference between $\mathrm{Residual}$ and Find is that in $\mathrm{Residual}$ all parameters depend on the questions words, where as in Find convolutional weights are the same for all questions, and only the element-wise multipliers γ vary based on the question. We note that the specific Find module we use in this work is slightly different from the one used in (Hu et al., 2017) in that it outputs a feature tensor, not just an attention map. This change was required in order to connect multiple Find modules in the same way as we connect multiple residual ones.

Based on the generic NMN model described above, we experiment with several specific architectures as shown in Figure 1. Each of the models uses $M = 3$ modules, which are connected and parametrized differently. In **NMN-Chain** modules form a sequential chain. Modules 1, 2 and 3 are parametrized based on the first object word, second object word and the relation word respectively, which is achieved by setting the attention α_1, α_2, α_3 to the corresponding one-hot vectors. We also experiment with giving the image features h_x as the right-hand side input to all 3 modules and call the resulting model NMN-Chain-Shortcut. **NMN-Tree** is similar to NMN-Chain in that the attention vectors are similarly hard-coded, but we change the connectivity between the modules to be tree-like. **Stochastic N2NMN** follows the N2NMN approach by Hu et al. (2017) for inducting layout. We treat the layout T as a stochastic latent variable. T is allowed to take two values: T_{tree} as in NMN-Tree, and T_{chain} as in NMN-Chain. We calculate the output probabilities by marginalizing out the layout i.e. probability of answer being "yes" is computed as $p(\text{yes}|x, q) = \sum_{T \in \{T_{tree}, T_{chain}\}} p(\text{yes}|T, x, q) p(T)$. Lastly, **Attention N2NMN** uses the N2NMN method for learning parametrization (Hu et al., 2017). Is structured just like NMN-Tree but has α^k computed as $\mathrm{softmax}(\tilde{\alpha}^k)$, where $\tilde{\alpha}^k$ is a trainable vector. We use Attention N2NMN only with the Find module because using it with the Find would involve a highly non-standard interpolation between convolutional weights.

4 EXPERIMENTS

In our experiments we aimed to: (a) understand which models are capable of exhibiting systematic generalization as required by SQOOP, and (b) understand whether it is possible to induce, in an end-to-end way, the successful architectural decisions that distinguish the best models.

All models share the same stem architecture which consists of 6 (8 for Relations Networks) layers of convolution, batch normalization and max pooling. The input to the stem is a $64 \times 64 \times 3$ image, and the feature dimension used throughout the stem is 64. Further details can be found in Appendix A. The code for all experiments will be released in the near future.

4.1 WHICH MODELS GENERALIZE BETTER?

We report the performance for all models on datasets of varying difficulty in Figure 3. 2 out of 4 models (Conv+LSTM and RelNet) were not able to learn to answer all SQOOP questions, no matter how high easy the split was (for high #rhs/lhs Conv+LSTM overfitted and RelNet did not train). The results of other two models, MAC and FiLM, are similar. Both models are clearly able to solve the SQOOP task, as suggested by their almost perfect $< 1\%$ error rate on the control #rhs/lhs=35 split, yet they struggle to generalize on splits with lower #rhs/lhs. In particular, we observe $13.67 \pm 9.97\%$ errors for MAC and a $34.73 \pm 4.61\%$ errors for FiLM on the hardest #rhs/lhs=1 split. For the splits of intermediate difficulty we saw the error rates of both models decreasing as we increased the #rhs/lhs ratio from 2 to 18. Interestingly, even with 18 #rhs/lhs some MAC and FiLM runs result in a test error rate of $\sim 2\%$. Given the simplicity and minimalism of SQOOP questions, we believe that these results should be considered a failure to pass the SQOOP test for both MAC and FiLM. That said, we note a difference in how exactly FiLM and MAC fail on #rhs/lhs=1: in several runs (3 out of 15) MAC exhibited a strong generalization performance ($\sim 0.5\%$ error rate) , whereas in all runs of FiLM the error rate was about 30%. We examined the successful MAC models and found that those that perform well converged to a successful setting of the control attention weights, that is the weights with which MAC units attend to questions words. In particular, MAC models that generalize strongly for each question would have a unit focusing strongly on X and a unit focusing strongly on Y. The reader is referred to Appendix B for more details on MAC results.

The results of the generic models that we described above should be contrasted with the performance of the modular and tree-structured NMN-Tree model. Both versions of this model, with Residual and Find modules, robustly solve all versions of our dataset, including the most challenging #rhs/lhs=1 split. As MAC was the strongest competitor of NMN-Tree across generic models, we have performed an ablation study for this model, in which we varied the number of modules and hidden units, as well as experimented with weight decay. These modifications have not resulted in any significant reduction of the gap between MAC and NMN-Tree. Interestingly, we found that using the default high number of MAC units, namely 12, had been helpful, possibly it made it more likely that some units are initialized to focus on X and Y words (see Appendix B for details).

4.2 WHAT IS ESSENTIAL TO STRONG GENERALIZATION OF NMN?

The NMN-Tree model differs from the generic models in that it is built from modules that are parametrized by question words directly, side-stepping a language encoder. Besides that, NMN-Tree is structured in a particular way, with the idea that modules 1 and 2 may learn to locate objects and module 3 can learn to reason about object locations independently of their identities. To understand which of the two differences is responsible for the superior generalization, we compare the performance of the NMN-Tree, NMN-Chain and NMN-Chain-Shortcut models (see Figure 1). The results in Figure 3 show that for both Find and Residual module architectures that we consider in this paper, using a tree layout is absolutely crucial (and sufficient) for generalization. In particular, NMN-Chain models perform barely above random chance, doing even worse than generic models on the #rhs/lhs=1 version of the dataset and dramatically failing even on the easiest #rhs/lhs=18 split. This is in stark contrast with NMN-Tree models that exhibits nearly perfect performance on the hardest #rhs/lhs=1 split. As a sanity check we trained NMN-Chain models on the vanilla #rhs/lhs=35 split. We found that NMN-Chain model has little difficulty to learn to answer SQOOP questions when it sees all of them at the training time, even though it shows very poor generalization in our other experiments. Interestingly, NMN-Chain-Shortcut performed much better than NMN-Chain and quite similarly to generic models. We find it remarkable that such a slight change in the model as adding the shortcut connections from image features h_x to the chain modules results in a drastic change in generalization performance. In an attempt to understand why NMN-Chain generalizes so poorly we compared the test set responses of the 5 NMN-Chain models trained on #rhs/lhs=1 split. Notably, there was very little agreement between predictions of these 5 runs (Fleiss $\kappa = 0.05$), suggesting that NMN-Chain performs rather randomly outside of the training set.

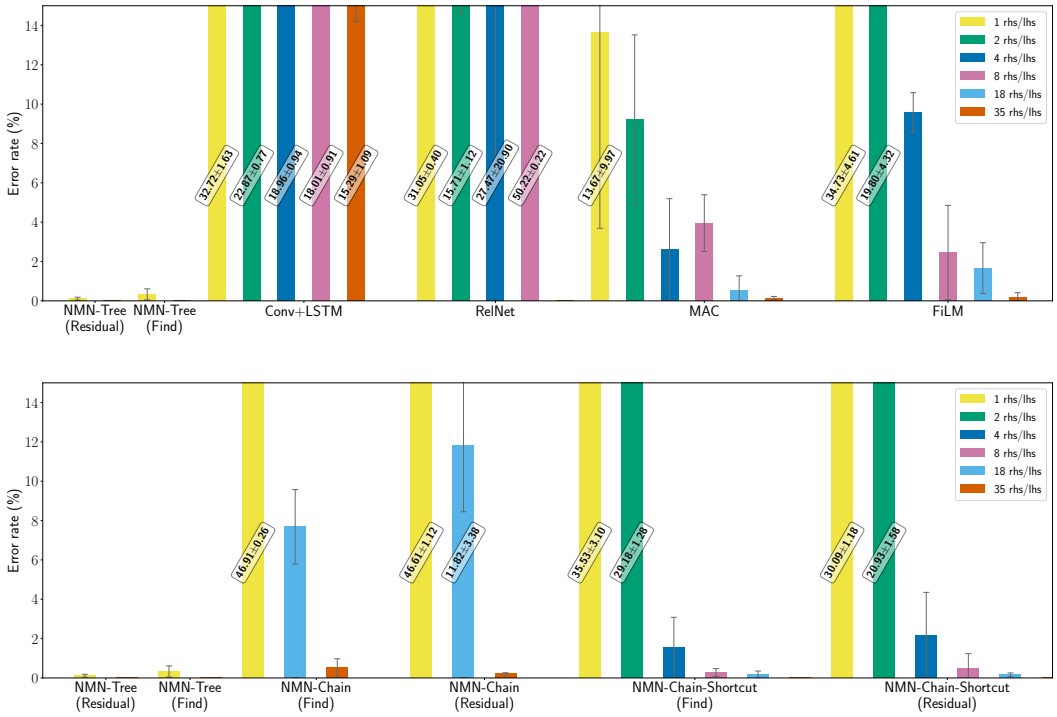


Figure 3: **Top:** Comparing the performance of generic models on datasets of varying difficulty (lower #rhs/lhs is more difficult). Note that NMN-Tree generalizes perfectly on the hardest #rhs/lhs=1 version of SQOOP, whereas MAC and FiLM fail to completely solve even the easiest #rhs/lhs=18 version. **Bottom:** Comparing NMNs with different layouts and modules. We can clearly observe the superior generalization of trees, poor generalization of chains and mediocre generalization of chains with shortcuts. Means and standard deviations after at least 5 runs are reported.

4.3 Can the Right Kind of NMN Be Induced?

The strong generalization of the NMN-Tree model is impressive, but a significant amount of prior knowledge about the task was required to come up with the successful *layout* and *parametrization* used in this model. We therefore investigate whether the amount of such prior knowledge can be reduced by fixing one of these structural aspects and inducing another.

4.3.1 Layout Induction

In our layout induction experiments, we use the Stochastic N2NMN model which treats the layout as a stochastic latent variable with two values (T_{tree} and T_{chain}, see Section 3.2 for details). We experiment with N2NMNs using both Find and Residual modules and report results with different initial conditions, $p_0(tree) \in 0.1, 0.5, 0.9$. We believe that the initial probability $p_0(tree) = 0.1$ should not be considered small, as in more challenging datasets the space of layouts would be exponentially large, and sampling the right layout in 10% of all cases could be considered a very lucky initialization. We repeat all experiments on #rhs/lhs=1 and on #rhs/lhs=18 splits, the former to study generalization, and the latter to control whether the failures on #rhs/lhs=1 are caused specifically by the difficulty of this split. The results (see Table 1) show that the success of layout induction (i.e. converging to a $p(tree)$ close to 0.9) depends in a complex way on all the factors that we considered in our experiments. The initialization has the most influence: models initialized with $p_0(tree) = 0.1$ typically do not converge to a tree (exception being experiments with Residual module on #rhs/lhs=18, in which 3 out of 5 runs converged to a solution with a high $p(tree)$). Likewise, models initialized with $p_0(tree) = 0.9$ always stay in a regime with a high $p(tree)$. In the intermediate setting of $p_0(tree) = 0.5$ we observe differences in behaviors for Residual and Find modules. In particular, N2NMN based on Residual modules stays spurious with $p(tree) = 0.5 \pm 0.08$ when #rhs/lhs=1, whereas N2NMN based on Find modules always converges to a tree.

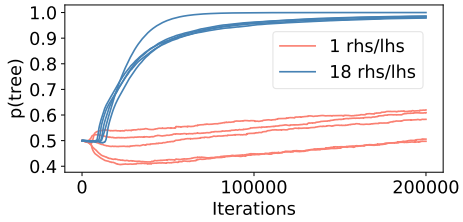


Figure 4: Learning dynamics of layout induction on 1 rhs/lhs and 18 rhs/lhs datasets using the Residual module with $p_0(tree) = 0.5$. All 5 runs do not learn to use the tree layout for 1 rhs/lhs, the very setting where the tree layout is necessary for generalization.

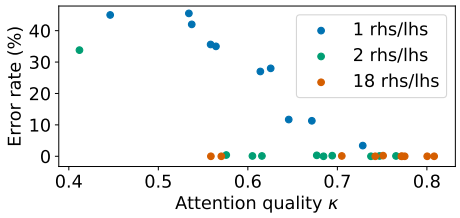


Figure 5: Attention quality κ vs accuracy for Attention N2NMN models trained on different #rhs/lhs splits. We can observe that generalization is strongly associated with high κ for #rhs/lhs=1, while for splits with 2 and 18 rhs/lhs blurry attention may be sufficient.

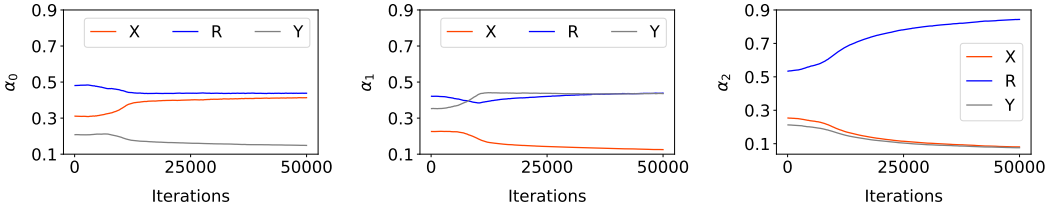


Figure 6: An example of how attention weights of modules 1 (**left**), 2 (**middle**), and 3 (**right**) evolve during training of an Attention N2NMN model on the 18 rhs/lhs version of SQOOP. Modules 1 and 2 learn to focus on different objects words, X and Y respectively in this example, but they also assign high weight to the relation word R. Module 3 learns to focus exclusively on R.

Table 1: Tree layout induction results for Stochastic N2NMNs using Residual and Find modules on 1 rhs/lhs and 18 rhs/lhs datasets. For each setting of $p_0(tree)$ we report results after 5 runs. $p_{200K}(tree)$ is the probability of using a tree layout after 200K training iterations.

module	#rhs/lhs	$p_0(tree)$	Test error rate (%)	Test loss	$p_{200K}(tree)$
Residual	1	0.1	31.89 ± 0.75	0.64 ± 0.03	0.08 ± 0.01
		0.5	1.64 ± 1.79	0.27 ± 0.04	0.56 ± 0.06
		0.9	0.16 ± 0.11	0.03 ± 0.01	0.96 ± 0.00
	18	0.1	3.99 ± 5.33	0.15 ± 0.06	0.59 ± 0.34
		0.5	0.19 ± 0.11	0.06 ± 0.02	0.99 ± 0.01
		0.9	0.12 ± 0.12	0.01 ± 0.00	1.00 ± 0.00
Find	1	0.1	47.54 ± 0.95	1.78 ± 0.47	0.00 ± 0.00
		0.5	0.78 ± 0.52	0.05 ± 0.04	0.94 ± 0.07
		0.9	0.41 ± 0.07	0.02 ± 0.00	1.00 ± 0.00
	18	0.1	5.11 ± 1.19	0.14 ± 0.03	0.02 ± 0.04
		0.5	0.17 ± 0.16	0.01 ± 0.01	1.00 ± 0.00
		0.9	0.11 ± 0.03	0.00 ± 0.00	1.00 ± 0.00

One counterintuitive result in Table 1 is that Stochastic N2NMNs with Residual modules that were trained with $p_0(tree) = 0.5$ and #rhs/lhs=1 make just $1.64 \pm 1.79\%$ errors on the generalization set despite being spurious mixtures between a tree and a chain. Our explanation for this phenomenon is as follows: when connected in a tree, modules of such spurious models generalize well, and when connected as a chain they generalize poorly. The output distribution of the whole model is thus a mixture of the mostly correct $p(y|T = T_{tree}, x, q)$ and mostly random $p(y|T = T_{chain}, x, q)$. We verified our reasoning by explicitly evaluating test accuracies for $p(y|T = T_{tree}, x, q)$ and $p(y|T = T_{chain}, x, q)$, and we found them to be around 99% and 60% respectively, confirming our hypothesis. As a result the predictions of the spurious models with $p(tree) \approx 0.5$ have lower confidence than those of sharp tree models, as indicated by the high log loss of 0.27 ± 0.04. We visualize the progress of structure induction for the Residual module with $p_0(tree) = 0.5$ in Figure 4 which shows how $p(tree)$ saturates to 1.0 for #rhs/lhs=18 and remains around 0.5 when #rhs/lhs=1.

Table 2: Parameterization induction results for 1,2,18 rhs/lhs datasets for Attention N2NMN. The model does not generalize well in the difficult 1 rhs/lhs setting. Results for MAC are presented for comparison. Means and standard deviations were estimated based on at least 10 runs.

Model	#rhs/lhs	Test error rate (%)	Test loss (%)
Attention N2NMN	1	27.19 ± 16.02	1.22 ± 0.71
Attention N2NMN	2	2.82 ± 8.18	0.14 ± 0.41
Attention N2NMN	18	0.16 ± 0.12	0.00 ± 0.00
MAC	1	13.67 ± 9.97	0.41 ± 0.32
MAC	2	9.21 ± 4.31	0.28 ± 0.15
MAC	18	0.53 ± 0.74	0.01 ± 0.02

4.3.2 Parametrization Induction

Next, we experiment with the Attention N2NMN model (see Section 3.2) in which the parametrization is learned for each module as an attention-weighted average of word embeddings. In these experiments, we fix the layout to be tree-like and sample the pre-softmax attention weights $\tilde{\alpha}$ from a uniform distribution $U[0; 1]$. As in the layout induction investigations, we experiment with several SQOOP splits, namely we try #rhs/lhs $\in \{1, 2, 18\}$. The results (reported in Table 2) show that Attention N2NMN fails dramatically on #rhs/lhs=1 but quickly catches up as soon as #rhs/lhs is increased to 2. Notably, 9 out of 10 runs on #rhs/lhs=2 resulted in almost perfect performance, and 1 run completely failed to generalize (26% error rate), resulting in a high 8.18% variance of the mean error rate. All 10 runs on the split with 18 rhs/lhs generalized flawlessly. We furthermore inspected the learned attention weights and found that for typical successful runs, module 3 focuses on the relation word, whereas modules 1 and 2 focus on different object words (see Figure 6) while still focusing on the relation word. To better understand the relationship between successful layout induction and generalization, we define an attention quality metric $\kappa = \min_{w \in \{X,Y\}} \max_{k \in 1,2} \alpha_{k,w}/(1 - \alpha_{k,R})$. Intuitively, κ is large when for each word $w \in X, Y$ there is a module i that focuses mostly on this word. The renormalization by $1/(1 - \alpha_{i,R})$ is necessary to factor out the amount of attention that modules 1 and 2 assign to the relation word. For the ground-truth parametrization that we use for NMN-Tree κ takes a value of 1, and if both modules 1 and 2 focus on X, completely ignoring Y, κ equals 0. The scatterplot of the test error rate versus κ (Figure 5) shows that for #rhs/lhs=1 high generalization is strongly associated with higher κ, meaning that it is indeed necessary to have different modules strongly focusing on different object words in order to generalize in this most challenging setting. Interestingly, for #rhs/lhs=2 we see a lot of cases where N2NMN generalizes well despite attention being rather spurious ($\kappa \approx 0.6$).

In order to put Attention N2NMN results in context were compare them to those of MAC (see Table 2). Such a comparison can be of interest because both models perform attention over the question. We note that Attention N2NMN succeeds in 9 out of 10 cases on the #rhs/lhs=2 split, much more often than 1 success out of 10 observed for MAC[1]. This result suggests that Attention N2NMNs retains some of the strong generalization potential of NMNs with hard-coded parametrization.

5 Related Work

Our study is inspired by prior substantial evidence that neural networks do not generalize on language understanding by performing systematic reasoning, but instead latch on to dataset-specific regularities. Agrawal et al. (2016) report how neural models exploit biases in a VQA dataset, e.g. responding "snow" to the question "what covers the ground" regardless of the image because "snow" is the most common answer to this question. Gururangan et al. (2018) report that many successes in natural language entailment are actually due to exploiting statistical biases as opposed to solving entailment, and that state-of-the-art systems are much less performant when tested on unbiased data. Jia & Liang (2017) demonstrate that seemingly state-of-the-art reading comprehension system can be misled by simply appending an unrelated sentence that resembles the question to the document.

Using synthetic VQA datasets to study grounded language understanding is a recent trend started by the CLEVR dataset (Johnson et al., 2016). CLEVR images are 3D-rendered and CLEVR ques-

[1]If we judge a run successful when the error rate is lower than $\tau = 1\%$, these success rates are different with a p-value of 0.001 according to the Fisher exact test. Same holds for any other threshold $\tau \in [1\%; 5\%]$.

tions are longer and more complex than ours, yet the color-shape generalization split that CLEVR includes arguably lacks a clear motivation. More closely related to our work is the ShapeWorld family of datasets by Kuhnle & Copestake (2017), that involves a number of VQA generalization tests. ShapeWorld only contains 10 different objects, making it insufficient for our study. Most closely related to our work is the recent study of generalization to long-tail questions about rare objects done by Bingham et al. (2018). They do not, however, consider as many models as we do and do not study the question of whether the best-performing models can be made end-to-end.

The key paradigm that we test in our experiments is Neural Module Networks (NMN). Andreas et al. (2016) introduced NMNs as a modular, structured VQA model where a fixed number of hand-crafted neural modules (such as `Find`, or `Compare`) are chosen and composed together in a layout determined by the dependency parse of question. Hu et al. (2017) and Johnson et al. (2017) followed up by making NMNs end-to-end, removing the non-differentiable parser. The former chose to keep the handcrafted modules and uses reinforcement learning to learn the layout and modules end-to-end. The latter used a ground truth module layout learned separately, and changes the hand-crafted modules for a generic ResNet block structure (He et al., 2016) for every module. Both Hu et al. (2017) and Johnson et al. (2017) reported that several thousands of ground-truth layouts are required to pretrain the layout predictor in order for their approaches to work. In a recent work Hu et al. (2018) attempt to remove the need for hard stochastic layout decisions, but training their models end-to-end from scratch performed substantially lower than best models on the CLEVR task.

6 Conclusion and Discussion

We have conducted a rigorous investigation of an important form of systematic generalization required for grounded language understanding: the ability to reason about all possible pairs of objects despite being trained on a small subset. Our results allow one to draw two important conclusions. For one, the intuitive appeal of modularity and structure in designing neural architectures for language understanding is now supported by our results, which show how a modular model consisting of general purpose residual blocks generalizes much better than a number of baselines, including architectures such as MAC, FiLM and RelNet that were designed specifically for visual reasoning. While this may seem unsurprising, to the best of our knowledge, the literature has lacked such a clear empirical evidence in favor of modular and structured networks before this work. Importantly, we have also shown how sensitive the high performance of the modular models is to the layout of modules, and how a tree-like structure generalizes much stronger than a typical chain of layers.

Our second key conclusion is that coming up with an end-to-end and/or soft version of modular models may be not sufficient for strong generalization. In the very setting where strong generalization is required, end-to-end methods may converge to a different, less compositional solution (e.g. a chain layout or blurred attention). This can be observed especially clearly in our NMN layout and parametrization induction experiments on the #rhs/lhs=1 version of SQOOP, but notably, strong initialization sensitivity of layout induction remains an issue even on the #rhs/lhs=18 split. This conclusion is relevant in the view of recent work in the direction of making NMNs more end-to-end (Suarez et al., 2018; Hu et al., 2018; Hudson & Manning, 2018; Gupta & Lewis, 2018). Our findings suggest that merely replacing hard-coded components with learnable counterparts may be insufficient, and that research on regularizers or priors that steer the learning towards more systematic solutions may be required. That said, our parametrization induction results on the #rhs/lhs=2 split are encouraging, as they show that compared to generic models, a weaker nudge (in the form of a richer training signal or a prior) towards systematicity may suffice for end-to-end NMNs.

While our investigation has been performed on a synthetic dataset, we believe that it is the real-world language understanding where our findings may be most relevant. It is possible to construct a synthetic dataset that is bias-free and that can only be solved if the model has understood the entirety of the dataset's language. It is, on the contrary, much harder to collect real-world datasets that do not permit highly dataset-specific solutions, as numerous dataset analysis papers of recent years have shown (see Section 5 for a review). We believe that approaches that can generalize strongly from imperfect and biased data will likely be required, and our experiments can be seen as a simulation of such a scenario. We hope, therefore, that our findings will inform researchers working on language understanding and provide them with a useful intuition about what facilitates strong generalization and what is likely to inhibit it.

REFERENCES

Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. Analyzing the Behavior of Visual Question Answering Models. pp. 1955–1960, January 2016. doi: 10.18653/v1/D16-1203.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural Module Networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL http://arxiv.org/abs/1511.02799.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the ICLR 2015*, 2015.

Eli Bingham, Piero Molino, Paul Szerlip, Obermeyer Fritz, and Goodman Noah. Chacterizing how Visual Question Answering scales with the world. 2018.

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71, 1988.

Alexander L Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. Differentiable programs with neural libraries. In *International Conference on Machine Learning*, pp. 1213–1222, 2017.

Yichen Gong, Heng Luo, and Jian Zhang. Natural Language Inference over Interaction Space. *arXiv:1709.04348 [cs]*, September 2017. URL http://arxiv.org/abs/1709.04348. arXiv: 1709.04348.

Nitish Gupta and Mike Lewis. Neural Compositional Denotational Semantics for Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2152–2161, Brussels, Belgium, 2018. Association for Computational Linguistics. URL http://aclweb.org/anthology/D18-1239.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation Artifacts in Natural Language Inference Data. *arXiv:1803.02324 [cs]*, March 2018. URL http://arxiv.org/abs/1803.02324. arXiv: 1803.02324.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to Reason: End-to-End Module Networks for Visual Question Answering. *arXiv:1704.05526 [cs]*, April 2017. URL http://arxiv.org/abs/1704.05526. arXiv: 1704.05526.

Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. Explainable neural computation via stack neural module networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 53–69, 2018.

Drew A. Hudson and Christopher D. Manning. Compositional Attention Networks for Machine Reasoning. February 2018. URL https://openreview.net/forum?id=S1Euwz-Rb.

Robin Jia and Percy Liang. Adversarial Examples for Evaluating Reading Comprehension Systems. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2021–2031, 2017. doi: 10.18653/v1/D17-1215. URL https://aclanthology.coli.uni-saarland.de/papers/D17-1215/d17-1215.

Yu Jiang, Vivek Natarajan, Xinlei Chen, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. Pythia v0.1: The winning entry to the vqa challenge 2018. https://github.com/facebookresearch/pythia, 2018.

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. *arXiv:1612.06890 [cs]*, December 2016. URL http://arxiv.org/abs/1612.06890. arXiv: 1612.06890.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Inferring and Executing Programs for Visual Reasoning. In *ICCV*, 2017. URL http://arxiv.org/abs/1705.03633.

Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, et al. Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 955–964. ACM, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

Alexander Kuhnle and Ann Copestake. ShapeWorld - A new test methodology for multimodal language understanding. *arXiv:1704.04517 [cs]*, April 2017. URL http://arxiv.org/abs/1704.04517. arXiv: 1704.04517.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*, 2018.

Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. In *Advances in neural information processing systems*, pp. 1682–1690, 2014.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual Reasoning with a General Conditioning Layer. In *In Proceedings of the 2017 AAAI Conference on Artificial Intelligence*, 2017. URL http://arxiv.org/abs/1709.07871.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *arXiv:1706.01427 [cs]*, June 2017. URL http://arxiv.org/abs/1706.01427. arXiv: 1706.01427.

Joseph Suarez, Justin Johnson, and Fei-Fei Li. DDRprog: A CLEVR Differentiable Dynamic Reasoning Programmer. *arXiv:1803.11361 [cs]*, March 2018. URL http://arxiv.org/abs/1803.11361. arXiv: 1803.11361.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pp. 3104–3112, 2014.

Wei Wang, Ming Yan, and Chen Wu. Multi-Granularity Hierarchical Attention Fusion Networks for Reading Comprehension and Question Answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1705–1714, Melbourne, Australia, 2018. Association for Computational Linguistics. URL http://aclweb.org/anthology/P18-1158.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, and others. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*, 2016.

A EXPERIMENT DETAILS

We trained all models by minimizing the cross entropy loss $\log p(y|x, q)$ on the training set, where $y \in \{\text{yes}, \text{no}\}$ is the correct answer, x is the image, q is the question. In all our experiments we used the Adam optimizer (Kingma & Ba, 2014) with hyperparameters $\alpha = 0.0001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-10}$. We continuously monitored validation set performance of all models during training, selected the best one and reported its performance on the test set. The number of training iterations for each model was selected in preliminary investigations based on our observations of how long it takes for different models to converge. This information, as well as other training details, can be found in Table 3.

Table 3: Training details for all models. The subsampling factor is the ratio between the original spatial dimensions of the input image and those of the representation produced by the stem. It is effectively equal to 2^k, where k is the number of 2x2 max-pooling operations in the stem.

model	stem layers	subsampling factor	iterations	batch size
FiLM	6	4	200000	64
MAC	6	4	100000	128
Conv+LSTM	6	4	200000	128
RelNet	8	8	500000	64
NMN (Residual)	6	4	50000	64
NMN (Find)	6	4	200000	64
Stochastic NMN (Residual)	6	4	200000	64
Stochastic NMN (Find)	6	4	200000	64
Attention NMN (Find)	6	4	50000	64

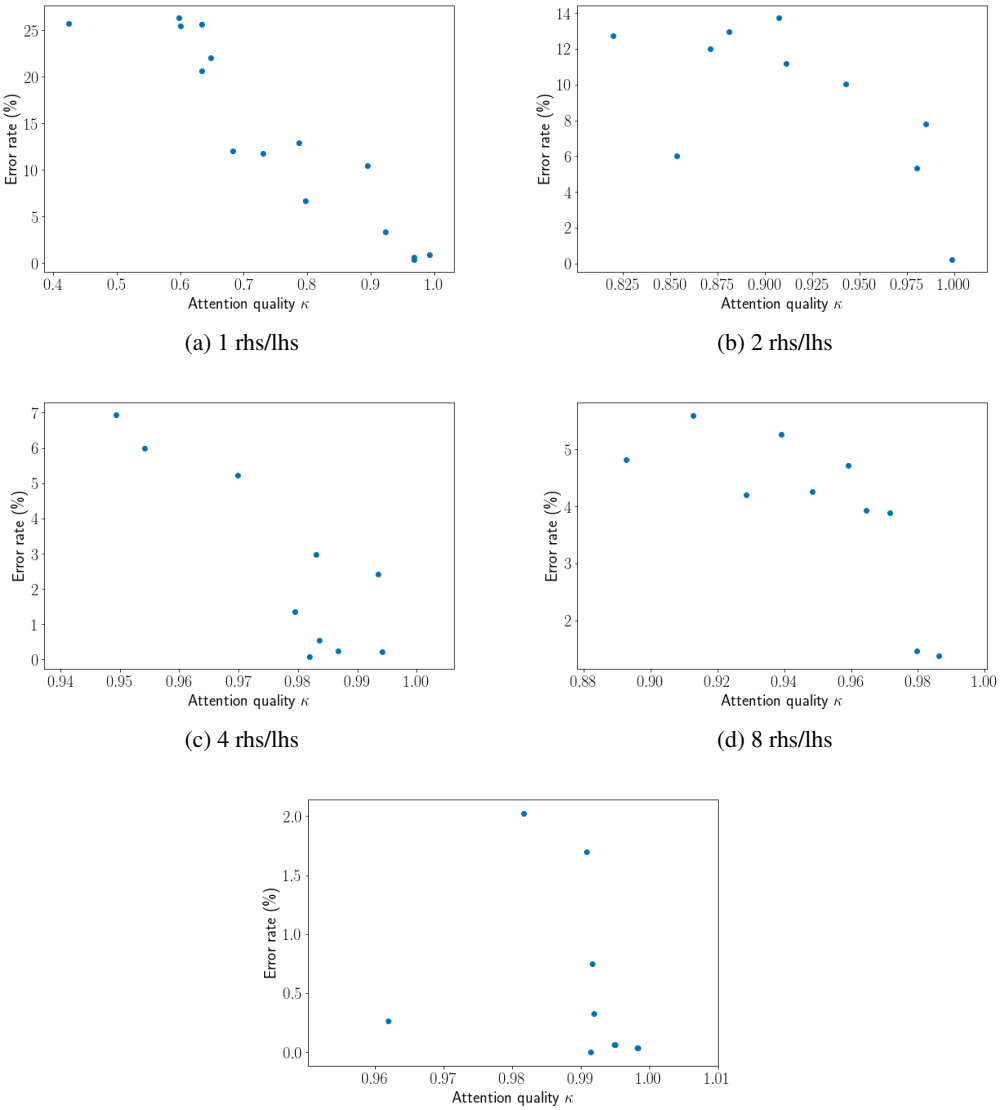
B ADDITIONAL RESULTS FOR MAC MODEL

We performed an ablation study in which we varied the number of MAC units, the model dimensionality and the level of weight decay for the MAC model. The results can be found in Table 4.

Table 4: Results of an ablation study for MAC. The default model has 12 MAC units of dimensionality 128 and uses no weight decay. For each experiment we report means and standard deviations based on 5 repetitions.

model	#rhs/lhs	train error rate (%)	test error rate (%)
default	1	0.17 ± 0.21	13.67 ± 9.97
1 unit	1	0.27 ± 0.35	28.67 ± 1.91
2 units	1	0.23 ± 0.13	24.28 ± 2.05
3 units	1	0.16 ± 0.15	26.47 ± 1.12
6 units	1	0.18 ± 0.17	20.84 ± 5.56
24 units	1	0.04 ± 0.05	9.11 ± 7.67
dim. 64	1	0.27 ± 0.33	23.61 ± 6.27
dim. 256	1	0.00 ± 0.00	4.62 ± 5.07
dim. 512	1	0.02 ± 0.04	8.37 ± 7.45
weight decay 0.00001	1	0.20 ± 0.23	19.21 ± 9.27
weight decay 0.0001	1	1.00 ± 0.54	31.19 ± 0.87
weight decay 0.001	1	40.55 ± 1.35	45.11 ± 0.74

We also perform qualitative investigations to understand the high variance in MAC's performance. In particular, we focus on control attention weights (c) for each run and aim to understand if runs that generalize have clear differences when compared to runs that failed. Interestingly, we observe that in successful runs each word $w \in \text{X, Y}$ has a unit that is strongly focused on it. To present our observations in quantitative terms, we plot attention quality $\kappa = \min_{w \in \{X,Y\}} \max_{k \in [1;12]} \alpha_{k,w}/(1-\alpha_{k,R})$, where α are control scores vs accuracy in Figure 7 for each run (see Section 4.3.2 for an explanation of κ). We can clearly see a strong positive correlation between κ and error rate.



(a) 1 rhs/lhs

(b) 2 rhs/lhs

(c) 4 rhs/lhs

(d) 8 rhs/lhs

(e) 18 rhs/lhs

Figure 7: Model test accuracy vs κ for various versions of SQOOP. All experiments are run 10 times with different random seeds. We can observe a clear correlation between κ and error rate for rhs/lhs =1,2,4. Also note that perfect generalization is always associated with κ close to 1.

Next, we experiment with a *hard-coded* variation of MAC. In this model, we use hard-coded control scores such that given a SQOOP question X R Y, the first half of all modules focuses on X while the second half focuses on Y. The relationship between MAC and hardcoded MAC is similar to that between NMN-Tree and end-to-end NMN with parameterization induction. However, this model has not performed as well as the successful runs of MAC. We hypothesize that this could be due to the interactions between the control scores and the visual attention part of the model.