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., 2016b; Gaunt et al., 2017). For example, in the Neural Module Network paradigm (NMN, Andreas et al. (2016a)), 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., 2016a) 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 CAN

(Hudson & Manning, 2018), and their generalization, to the best of our knowledge, has not been a subject of a focused study.

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.

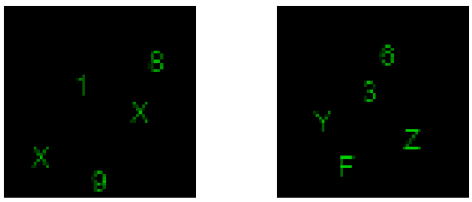
Our first finding is that NMNs do generalize better than other neural models when an appropriate choice of layout and parametrization is made. We then investigate which factors contribute to improved generalization performance and find that using a tree layout, as opposed to a chain layout, is crucial 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) . We show how such end-to-end approaches often fail to find the right structural settings and instead prefer a wrong chain layout or spurious parametrization and do not generalize better than the generic models. 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 VQA MODELS

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 $36 \cdot 36 \cdot 4 = 5184$ possible unique questions. 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 questions are illustrated in Figure 1.

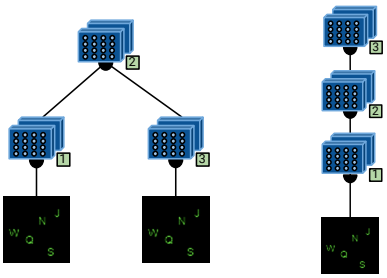
Our goal is to discover which models can correctly answer questions about all $36 \cdot 36$ possible object pairs in the SQOOP dataset after having been trained on only a subset. Therefore, we build training sets containing $36 \cdot 4 \cdot k$ unique questions such that for every left-hand-side (LHS) object



a: Is there a 9 right of a X? **True**

b: Is there a 3 right of a 6? **False**

Figure 1: The SQOOP dataset tests whether models can learn to generalize simple spatial relations to all object pairs while being trained on a small subset of them. We show a positive (left) and negative (right) example of the constructed dataset.



a: NMN-tree

b: NMN-chain

Figure 2: The different NMN layouts for the question "is there a W left of J".

X, we randomly sample k different right-hand-side (RHS) objects Y_1, Y_2, ..., Y_k. We will refer to k as the *#rhs/lhs parameter* of the dataset. Our test set is generated from the remaining $36 \cdot 4 \cdot (36 - k)$ questions. We generate training and test sets for several values of rhs/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 datasets of 8 rhs/lhs we generate $\approx \frac{10^6}{36 \cdot 4 \cdot 8}$ different images per question. Since our objective is to test language generalization, we minimize any distribution mismatch between images from train and test sets by ensuring that the background object distribution is matched in train and test.

3 MODELS

A great variety of VQA models have been recently proposed in the literature, among which 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., 2016b), 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 Networks (RelNet), and Compositional Attention Networks (CAN). 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 $h_{q\,\mathrm{x}}$.

CAN networks of Hudson & Manning (2018) produces $h_{q\,\mathrm{x}}$ by repeatedly applying a Memory-Attention-Control (MAC) cell that is conditioned on the question through an attention mechanism. The CAN 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., 2016b) 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 Residual module from Johnson et al. (2017) with very minor modifications - we use 64 dimensional CNNs in our Residual blocks since our dataset consists of 64 $\times$ 64 images. The equations for the 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 Residual and Find is that in 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.

Based on the generic NMN model described above, we experiment with several specific architectures. Each of the models uses $M = 3$ modules, which are connected and parametrized differently.

NMN-Chain has modules form a sequential chain as shown in Figure 1b. 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.

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, as shown in Figure 1a.

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 $p(\text{yes}|x, q) = \sum_{T \in \{T_{tree}, T_{chain}\}} p(\text{yes}|T, x, q)p(T)$.

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 $\text{softmax}(\tilde{\alpha}^k)$, 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.

4.1 Architecture Details and Hyperparameters

All models share the same stem architecture which is a CNN based architecture of 6 layers. Each layer is a Conv $\rightarrow$ BatchNorm $\rightarrow$ ReLU with a MaxPool after layers 1 and 3. The input to the stem is a 64 $\times$ 64 $\times$ 3 image, and the feature dimension used throughout the stem is 64.

4.2 Which Models Generalize Better?

We report the performance for a number models on datasets of varying difficulty in Table 1. These results show that although in principle generic models are capable of answering all SQOOP questions, they aren't able to generalize well on less redundant and more challenging versions of our

Table 1: Comparing the performance of generic models to the structured NMN-Tree model on datasets of varying difficulty (lower #rhs/lhs is more difficult). Confidence intervals are reported for experiments where we want to make the case that an approach does not work well.

model	#rhs/lhs	train. acc (%)	test acc. (%)	details
Conv+LSTM	1	97.9	64.4 ± 1.8	conf. interval after 5 runs
RelNet	1	95.6	63.1 ± 1.0	conf. interval after 5 runs
FiLM	1	100	66.6 ± 2.5	conf. interval after 5 runs
FiLM	2	100	85.4	best of 3 runs
FiLM	4	97.9	94.6	best of 3 runs
FiLM	8	98.6	97.4	best of 3 runs
MAC	1	99.5	72.6 ± 3.4	conf. interval after 5 runs
MAC	2	99.7	77.9	best of 3 runs
MAC	4	99.2	94.6	best of 3 runs
MAC	8	100	100	best of 3 runs
NMN-Tree (Residual)	1	100	100.0 ± 0.0	all 5 runs

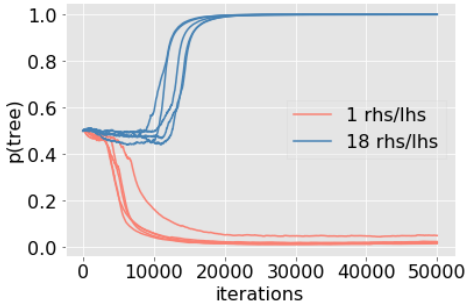


Figure 3: Learning dynamics of layout induction on 1 rhs/lhs and 18 rhs/lhs datasets using the Residual module with $p_0(tree) = 0.5$. All 5 runs of the model do not learn to use the tree layout for 1 rhs/lhs, the very setting where the tree layout is necessary for generalization.

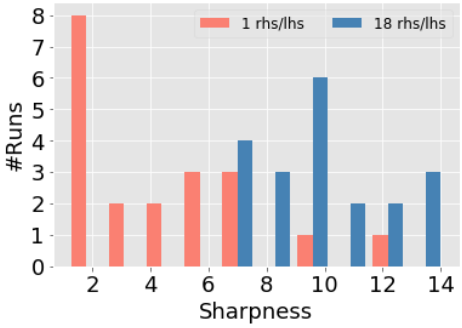


Figure 4: Histogram of sharpness (ρ) values for attention weights induced on the 1 rhs/lhs and 18 rhs/lhs datasets. We can observe that the attention is much sharper for 18 rhs/lhs.

dataset with #rhs/lhs=1 or #rhs/lhs=2. On the contrary, the modular and tree-structured NMN-Tree model easily solves all versions of our dataset.

4.3 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

Table 2: Comparing different NMN performance on the dataset with 1 #rhs/lhs. The module used in each model is indicated in the brackets. We can clearly observe that chains fail to generalize for this setting, whereas trees are successful.

model	train. acc (%)	test acc. (%)
NMN-Tree (Residual)	100	100.0 ± 0.0
NMN-Tree (Find)	100.0	99.7 ± 0.3
NMN-Chain (Find)	99.2	51.4 ± 2.8
NMN-Chain-XYR (Residual)	100	51.6 ± 1.6
NMN-Chain-XRY (Residual)	99.7	54.1 ± 1.7
NMN-Chain-RXY (Residual)	98.7	50.5 ± 0.9

Table 3: Tree layout induction successes for Stochastic N2NMNs using Residual modules and Find modules. For each setting of $p_0(tree)$, we report results after 5 runs on 1 rhs/lhs and 18 rhs/lhs datasets.

(a) Residual modules

#rhs/lhs	$p_0(tree)$	test acc. (%)	$p_{50K}(tree)$
1	0.1	52.7 ± 2.2	0.003
	0.5	57.0 ± 4.4	0.026
	0.9	99.9 ± 0.1	0.997
18	0.1	100.0 ± 0.0	0.999
	0.5	97.7 ± 5.1	0.999
	0.9	99.1 ± 2.3	0.999

(b) Find modules

#rhs/lhs	$p_0(tree)$	test acc. (%)	$p_{50K}(tree)$
1	0.1	51.2 ± 2.9	0.00
	0.5	93.2 ± 7.1	0.999
	0.9	95.9 ± 1.6	0.999
18	0.1	78.6 ± 20.7	0.2
	0.5	91.6 ± 6.5	0.999
	0.9	97.3 ± 3.4	0.999

Table 4: Parameterization induction results for 1 rhs/lhs and 18 rhs/lhs datasets for Attention N2NMN model. Though the model fits the training set, it doesn't fully generalize in the more difficult 1 rhs/lhs setting.

#rhs/lhs	train acc.	test acc.
1	97.8 ± 3.11	83.79 ± 9.6
18	98.54 ± 1.0	99.2 ± 1.6

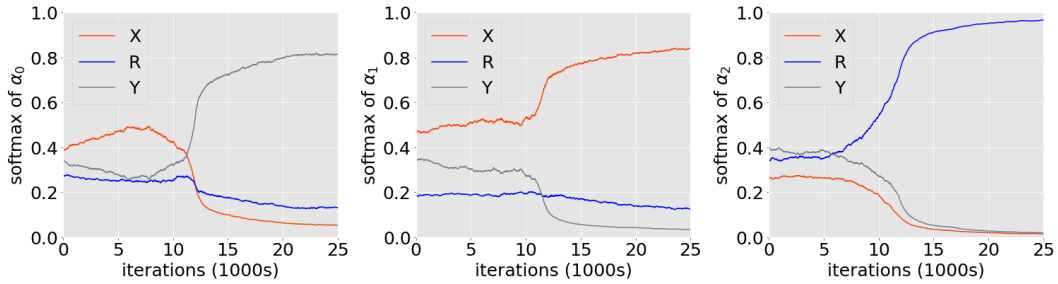
and module 3 can learn to reason about object locations independently of their identities. To understand which of the two differences is responsible for the superior generalization, we compare the performance of the NMN-Tree and NMN-Chain models (see Figure 2). The results in Table 2 show that for both module architectures that we consider in this paper, using a tree layout is absolutely crucial (and sufficient) for generalization. Notably, NMN-Chain models perform barely above random chance, performing even worse than generic models on the #rhs/lhs=1 version of the dataset. This is in stark contrast with NMN-Chain models that exhibit nearly perfect performance.

4.4 Can the Right Kind of NMN Be Induced?

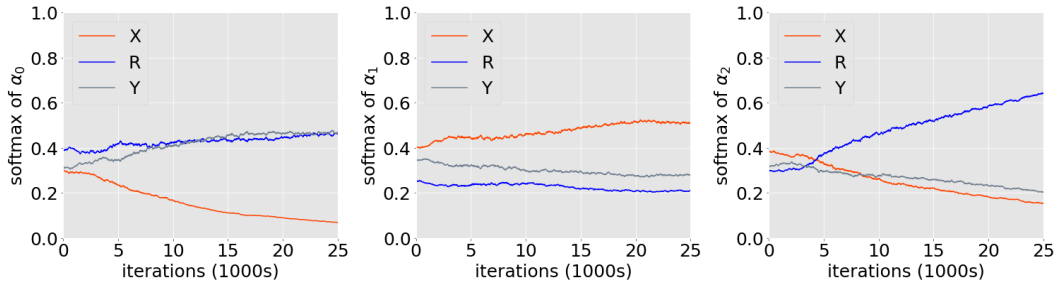
While the strong generalization of the NMN-Tree model is impressive, it is in some sense unsurprising because both the *layout* as well as the *parametrization* of this model encode a significant amount of prior knowledge about the task. We therefore investigate whether the amount of such prior knowledge can be reduced by trying to fix one of these structural aspects and induce another.

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 diverse initial conditions, $p_0(tree) = 0.1, 0.5, 0.9$. The results obtained on the #rhs/lhs=1 dataset (reported in Table 3) show that the correct tree layout was not induced for $p_0(tree) = 0.1$ and $p_0(tree) = 0.5$ when we use the Residual module and for $p_0(tree) = 0.1$ and we use the Find module. To put these results in context, we also run similar experiments on an easy to generalize #rhs/lhs=18 version. This time the NMN with the Residual module preferred the tree layout for $p_0(tree)$. It is notable, however, that in the setting with #rhs/lhs=1 where the correct choice of the layout is the *only* way to generalize, only a very lucky initialization $p_0(tree) = 0.9$ resulted in successful layout induction for the Residual module. Finally, we note that layout induction is more successful with Find compared to Residual, which casts light on the brittle nature of the induction which is highly dependent on the exact architecture of the module employed.

Parameterization Induction: We furthermore experimented with the Attention N2NMN model (see Section 3.2) in which the parametrization is learned for each module as a 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 investigation, we experiment with #rhs/lhs=1 and #rhs/lhs=18, the latter being a sanity-check of our implementation. The results, shown in Table-4, suggest that the model did not always find the attention settings that lead to generalization on the challenging #rhs/lhs=1 split (83.8% test accuracy). That should be contrasted with the close-to-perfect accuracy of the model that was trained on #rhs/lhs=18 version of the task (99.2%), suggesting that the parametrization induction did not



(a) 18 rhs/lhs



(b) 1 rhs/lhs

Figure 5: All three modules' attention weights for parametrization of the three question words for (a) 18 rhs/lhs and (b) 1 rhs/lhs version of SQOOP. The model learns to disentangle between X and Y much better with more rhs/lhs.

work due to the difficulty of our #rhs/lhs=1 split. The numbers reported above are averages over 10 runs and the difference is statistically significant according to a two-sided T-test ($p = 0.007$).

We visualize the progress of structure induction for the Residual module with $p_0(tree) = 0.5$ in Figure 3. The figure shows $p(tree)$ saturates to 0.0 or 1.0 eventually in #rhs/lhs=1 and #rhs/lhs=18 settings respectively. Figure 5 shows how attention weights evolve for an Attention N2NMN model in the same context. It is notable that unlike in the gold-standard NMN-Tree model, the relation word is mixed with the object words for modules 1 and 2. We also noticed that the model did not learn to focus modules 1 and 2 on different words in the #rhs/lhs=1 setting (Figure 5b) as sharply as it did in #rhs/lhs=18 (Figure-5a). To substantiate this observation with quantitative results, we compute a sharpness ratio $\rho = \max(\alpha^{k,X}, \alpha^{k,Y})/\min(\alpha^{k,X}, \alpha^{k,Y})$ for modules $k = 1$ and $k = 2$ for each of the 20 modules that we have trained. One can observe from the histogram in Figure 4 that attention weights learnt on #rhs/lhs=1 are generally blurry, with ρ being less than 2 for 8 modules out of 20.

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 questions 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. We could not use ShapeWorld for our investigation because our study required a large number of different objects and ShapeWorld only contains 10. 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. (2016a) 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 concurrent work by (Hu et al., 2018), an attempt has been made to remove the need for hard stochastic layout decisions, but end-to-end training from scratch of such models has performed substantially lower than best models on the above-mentioned 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 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, because in the very setting where strong generalization is required, end-to-end methods may find a different, less compositional solution (e.g. a chain layout or blurred attention). This conclusion is relevant in the view of recent work done in the direction of making Neural Module Networks more end-to-end (Suarez et al., 2018; Hu et al., 2018; Hudson & Manning, 2018). Our findings suggest that merely replacing hard-coded components with learnable counterparts may be insufficient, and that research on finding regularizers or priors that steer the learning towards more systematic solutions may be required.

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. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 39–48, 2016a.

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)*, 2016b. 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.

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.

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 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. February 2018. URL https://openreview.net/forum?id=HypkN9yRW.

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.