Abstract

Compositional generalization is a basic mechanism in human language learning, but current neural networks lack such ability. In this paper, we conduct fundamental research for encoding compositionality in neural networks. Conventional methods use a single representation for the input sentence, making it hard to apply prior knowledge of compositionality. In contrast, our approach leverages such knowledge with two representations, one generating attention maps, and the other mapping attended input words to output symbols. We reduce the entropy in each representation to improve generalization. Our experiments demonstrate significant improvements over the conventional methods in five NLP tasks including instruction learning and machine translation. In the SCAN domain, it boosts accuracies from 14.0% to 98.8% in Jump task, and from 92.0% to 99.7% in TurnLeft task. It also beats human performance on a few-shot learning task. We hope the proposed approach can help ease future research towards human-level compositional language learning. Source code is available online.

1 Introduction

Humans learn language in a flexible and efficient way by leveraging systematic compositionality, the algebraic capacity to understand and produce large amount of novel combinations from known components (Chomsky, 1957; Montague, 1970). For example, if a person knows how to “step”, “step twice” and “jump”, then it is natural for the person to know how to “jump twice” (Figure 1). This compositional generalization is critical in human cognition (Minsky, 1986; Lake et al., 2017), and it helps humans learn language from a limited amount of data, and extend to unseen sentences.

Figure 1: Problem illustration. If a person knows how to “step”, “step twice” and “jump”, then the person should know how to “jump twice”. Although this is straightforward for humans, it remains an open problem to equip machines with such ability.

In recent years, deep neural networks have made many breakthroughs in various problems (LeCun et al., 2015; Krizhevsky et al., 2012; Yu and Deng, 2012; He et al., 2016; Wu et al., 2016). However, there have been critiques that neural networks do not have compositional generalization ability (Fodor and Pylyshyn, 1988; Marcus, 1998; Fodor and Lepore, 2002; Marcus, 2003; Calvo and Symons, 2014).

Our observation is that conventional methods use a single representation for the input sentence, which makes it hard to apply prior knowledge of compositionality. In contrast, our approach leverages such knowledge with two representations, one generating attention maps, and the other mapping attended input words to output symbols. We reduce each of their entropies to improve generalization. This mechanism equips the proposed approach with the ability to understand and produce novel combinations of known words and to achieve compositional generalization.

In this paper, we focus on compositional gener-
Table 1: Examples of SCAN input commands (left) and output action sequences (right). Upper section is for training, and lower section is for testing. These tasks are difficult because “jump” (or “turn left”) does not appear with other words in training.

(a) Jump task. In training, “jump” only appears as a single command. In test, it appears with other words.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump run after run left</td>
<td>JUMP LTURN RUN RUN</td>
</tr>
<tr>
<td>look left twice and look opposite right</td>
<td>LTURN LOOK LTURN LTTURN RTURN RTURN LOOK</td>
</tr>
<tr>
<td>jump twice after look</td>
<td>LOOK JUMP JUMP JUMP LTURN</td>
</tr>
<tr>
<td>turn left after jump twice</td>
<td>LTURN JUMP JUMP JUMP LTURN</td>
</tr>
<tr>
<td>jump right twice after jump left twice</td>
<td>LTURN JUMP LTURN JUMP RTURN JUMP RTURN</td>
</tr>
</tbody>
</table>

(b) TurnLeft task. In training, “turn left” only appears as a single command. In test, it appears with other words.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>turn left run thrice and jump right</td>
<td>LTURN RUN RUN RUN RTURN JUMP</td>
</tr>
<tr>
<td>look left thrice after run left twice</td>
<td>LTURN RUN LTURN RUN LTURN LTURN LOOK LTURN</td>
</tr>
<tr>
<td>look twice and turn left twice</td>
<td>LOOK LOOK LTURN LTURN LTURN</td>
</tr>
<tr>
<td>turn left thrice and turn left</td>
<td>LTURN LTURN LTURN LTURN RTURN RTURN</td>
</tr>
<tr>
<td>turn left twice after look opposite right twice</td>
<td>RTURN RTURN LOOK RTURN RTURN LOOK LTURN LTURN</td>
</tr>
</tbody>
</table>

Besides Primitive tasks, we experiment on more tasks in SCAN and other datasets for instruction learning and machine translation. We also introduce an extension of SCAN dataset, dubbed as SCAN-ADJ, which covers additional compositional challenges. Our experiments demonstrate that the proposed method achieves significant improvements over the conventional methods. In the SCAN domain, it boosts accuracies from 14.0% to 98.8% in Jump task, and from 92.0% to 99.7% in TurnLeft task. It also beats human performance on a few-shot learning task.

2 Approach

In compositional generalization tasks, we consider both input $X$ and output $Y$ have multiple components that are not labeled in data. Suppose input has two components $X_1$ and $X_2$, and output has two components $Y_1$ and $Y_2$, then we can define compositional generalization probabilistically as follows. In training,

\[ P(X_1) > 0 \text{ and } P(X_2) > 0, \]
\[ P(X_1, X_2) = 0, \]
\[ P(Y_1 | X_1) \text{ and } P(Y_2 | X_2) \text{ are high.} \]

In test,

\[ P(X_1, X_2) > 0, \]
\[ P(Y_1, Y_2 | X_1, X_2) \text{ is predicted high.} \]
Figure 2: Illustration of network structure and flowchart for the proposed approach. Input is a sequence of words. Output is a sequence of actions. An input sentence is converted to primitive representation (left) and function representation (right). In training, we reduce entropy of two representations. Function representation is used to generate a sequence of attention maps. Each attention map is used to select an embedding from primitive representation to output an action.

We expect this is possible when we have prior knowledge that $Y_1$ depends only on $X_1$ and $Y_2$ depends only on $X_2$.

\[
P(Y_1|X_1, X_2, Y_2) = P(Y_1|X_1), \]
\[
P(Y_2|X_1, X_2, Y_1) = P(Y_2|X_2).
\]

Therefore, with chain rule,

\[
P(Y_1, Y_2|X_1, X_2) = P(Y_1|X_1)P(Y_2|X_2).
\]

Based on these arguments, we design the algorithm with the following objectives:

(i) Learn two representations for input.

(ii) Reduce entropy in each representation.

(iii) Make output action type depend only on one representation, and output action order depend only on the other.

Figure 2 shows an illustration of flowchart for the algorithm design (see Algorithm 1 for details). We describe it further here.

2.1 Input and Output

Input $x$ contains a sequence of $n$ words, where each input word is from an input vocabulary of size $U$. Output $y$ contains a sequence of $m$ action symbols, where each output symbol is from an output vocabulary of size $V$. Both vocabularies contain an end-of-sentence symbol which appears at the end of $x$ and $y$, respectively. The model output $\hat{y}$ is a prediction for $y$. Both input words and
We concatenate Algorithm 1 Proposed approach. Decoder in Seq2seq architecture is autoregressive model here.

Require: $V$: output vocabulary size $k_p, k_f$: embedding sizes $\alpha$: 1 for training, 0 for inference
Seq2seq: seq2seq architecture

Input: $x$: sequence of one-hot representations
Output: $\hat{y}$: sequence of distributions

1: for $i = 1, \ldots, n$ do
2: $p_i = \text{Emb}_p(x_i) \in \mathbb{R}^{k_p}$
3: $f_i = \text{Emb}_f(x_i) \in \mathbb{R}^{k_f}$
4: end for
5: $p' = p + \alpha \epsilon_p \in \mathbb{R}^{n \times k_p}, \epsilon_p \sim \mathcal{N}(0, I)$
6: $f' = f + \alpha \epsilon_f \in \mathbb{R}^{n \times k_f}, \epsilon_f \sim \mathcal{N}(0, I)$
7: decoder = Seq2seq($f'$)
8: for $j = 1, \ldots, \text{max length}$ do
9: $b_j = \text{decoder}.\text{next}() \in \mathbb{R}^{n}$
10: $a_j = \text{Softmax}(b_j) \in \mathbb{R}^{n}$
11: $v_j = a_j p' \in \mathbb{R}^{k_p}$
12: $l_j = f_{\text{predict}}(v_j) \in \mathbb{R}^V$
13: $y_j = \text{Softmax}(l_j) \in (0, 1)^V$
14: break if argmax $y_j$ is <EOS>
15: end for

output symbols are in one-hot representation, i.e.,

$x = [x_1, \ldots, x_n] \in \{0, 1\}^{U \times n}$,
$y = [y_1, \ldots, y_m] \in \{0, 1\}^{V \times m}$.

2.2 Primitive Representation and Function Representation

To achieve objective (i), we use two representations for an input sentence. For each word, we use two word embeddings for primitive and functional information.

$p_i = \text{Emb}_p(x_i) \in \mathbb{R}^{k_p}$,
$f_i = \text{Emb}_f(x_i) \in \mathbb{R}^{k_f}$.

We concatenate $p_i$ and $f_i$ to form the primitive representation and functional representation for the entire input sequence, i.e.,

$p = [p_1, \ldots, p_n] \in \mathbb{R}^{k_p \times n}$,
$f = [f_1, \ldots, f_n] \in \mathbb{R}^{k_f \times n}$.

2.3 Entropy Regularization

To achieve objective (ii), we reduce the amount of information, or entropy of the representations. We do this by regularizing the $L_2$ norm of the representations, and then adding noise to them. This reduces channel capacities, and thus entropy for the representations.

$p' = p + \alpha \epsilon_p \in \mathbb{R}^{k_p \times n}, \epsilon_p \sim \mathcal{N}(0, I)$,
$f' = f + \alpha \epsilon_f \in \mathbb{R}^{k_f \times n}, \epsilon_f \sim \mathcal{N}(0, I)$,
$L_{\text{regularize}} = L_2(p) + L_2(f)$.

Here, $\alpha$ is the weight for noise. Noise is helpful in training, but is not necessary in inference. Therefore, we set $\alpha$ as a hyper-parameter for training and $\alpha = 0$ for inference.

2.4 Combine Representations and Generate Output

To achieve objective (iii), we want to make the output action types depend only on primitive representation, and the output action order depend only on function representation. Attention mechanism is a good fit for this purpose. We generate a sequence of attention maps from function representation, and use each attention map to get the weighted average of primitive representation to predict the corresponding action.

We use attention mechanism with sequence-to-sequence architectures to aggregate representations in the following ways: (1) Instead of outputting a sequence of logits on vocabulary, we emit a sequence of logits for attention maps on input positions (see Figure 2 for an example). (2) During decoding for autoregressive models, we do not use the previous output symbol as input for the next symbol prediction, because output symbols contain information for primitives. We still use the last decoder hidden state as input. Non-autoregressive models should not have this problem. (3) We end decoding when the last output is end-of-sentence symbol.

More specifically, we feed the function representation, $f'$, to a sequence-to-sequence neural network for decoding. At each step $j$, we generate $b_j \in \mathbb{R}^n$ from the decoder, and use Softmax to obtain the attention map, $a_j$. With attention maps, we extract primitives and generate output symbols. To do that, we compute the weighted average, $v_j$, on noised primitive representations, $p'$, with attention, $a_j$. Then we feed it to a one-layer fully connected network, $f_{predict}$. We use Softmax to compute the output distribution, $\hat{y}_j$. The decoding ends when
argmax $\hat{y}_j$ is end-of-sentence symbol.

$$a_j = \text{Softmax}(b_j) \in \mathbb{R}^n,$$
$$v_j = a_j p' \in \mathbb{R}^k,$$
$$l_j = f_{\text{predict}}(v_j) \in \mathbb{R}^V,$$
$$\hat{y}_j = \text{Softmax}(l_j) \in (0, 1)^V.$$  

2.5 Loss

We use the cross entropy of $y$ and $\hat{y}$ as the prediction loss, $L_{\text{prediction}}$, and the final loss, $L$, is the combination of prediction loss and entropy regularization loss (Section 2.3). $\lambda$ is the regularization weight.

$$L_{\text{prediction}} = \sum_{j=1}^m \text{CrossEntropy}(y_j, \hat{y}_j),$$
$$L = L_{\text{prediction}} + \lambda L_{\text{regularize}}.$$  

3 Experiments

We ran experiments on instruction learning and machine translation tasks. Following Lake and Baroni (2018), we used sequence-level accuracy as the evaluation metric. A prediction is considered correct if and only if it perfectly matches the reference. We ran all experiments five times with different random seeds for the proposed approach, and report mean and standard deviation (note that their sum may exceed 100%). The Appendix contains details of models and experiments.

3.1 SCAN Primitive Tasks

We used the SCAN dataset (Lake and Baroni, 2018), and targeted the Jump and TurnLeft tasks. For the Jump and TurnLeft tasks, “jump” and “turn left” appear only as a single command in their respective training sets, but they appear with other words at test time (see Table 1a and Table 1b, respectively).

We applied the same configuration for all tasks (see Appendix A for details). The result in Table 2 shows that the proposed method boosts accuracies from 14.0% to 98.8% in Jump task, and from 92.0% to 99.7% in TurnLeft task.

<table>
<thead>
<tr>
<th>Method</th>
<th>Jump</th>
<th>TurnLeft</th>
</tr>
</thead>
<tbody>
<tr>
<td>L&amp;B best overall</td>
<td>0.1</td>
<td>90.0</td>
</tr>
<tr>
<td>L&amp;B best</td>
<td>1.2</td>
<td>90.3</td>
</tr>
<tr>
<td>RNN +Attn-Dep</td>
<td>2.7 ± 1.7</td>
<td>92.0 ± 5.8</td>
</tr>
<tr>
<td>GRU +Attn</td>
<td>12.5 ± 6.6</td>
<td>59.1 ± 16.8</td>
</tr>
<tr>
<td>Ptr-Bahdanau</td>
<td>3.3 ± 0.7</td>
<td>91.7 ± 3.6</td>
</tr>
<tr>
<td>Ptr-Luong</td>
<td>14.0 ± 2.8</td>
<td>66.0 ± 5.6</td>
</tr>
<tr>
<td>Proposed</td>
<td>98.8 ± 1.4</td>
<td>99.7 ± 0.4</td>
</tr>
</tbody>
</table>

Table 2: Test accuracy (mean ± std %) for SCAN Primitive tasks. L&B is from Lake and Baroni (2018). RNN and GRU are best models for each task from Bastings et al. (2018). Ptr-Bahdanau and Ptr-Luong are from Kliegl and Xu (2018). The proposed method achieves significantly higher performance than baselines.

3.2 SCAN Template-matching

We also experimented with the SCAN template-matching task. The task and dataset descriptions can be found in Loula et al. (2018b). There are four tasks in this dataset: jump around right, primitive right, primitive opposite right and primitive around right. Each requires compositional generalization. Please see Appendix B for more details.

Table 3 summarizes the results on SCAN template-matching tasks. The baseline methods are from Loula et al. (2018b). The proposed method achieves significantly higher performance than baselines in all tasks. This experiment demonstrates that the proposed method has better compositional generalization related to template matching.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Baseline</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump around</td>
<td>98.4 ± 0.5</td>
<td>100.0 ± 0.0</td>
</tr>
<tr>
<td>Prim.</td>
<td>23.5 ± 8.1</td>
<td>99.7 ± 0.5</td>
</tr>
<tr>
<td>Prim. opposite</td>
<td>47.6 ± 17.7</td>
<td>89.3 ± 5.5</td>
</tr>
<tr>
<td>Prim. around</td>
<td>2.5 ± 2.7</td>
<td>83.2 ± 13.2</td>
</tr>
</tbody>
</table>

Table 3: Test accuracy (mean ± std %) for SCAN template-matching tasks. The baseline methods are from Loula et al. (2018b). The proposed method achieves significantly higher performance than baselines.

3.3 Primitive and Functional Information Exist in One Word

One limitation of SCAN dataset is that each word contains either primitive or functional information, but not both. For example, “jump” contains only primitive information, but no functional information. On the other hand, “twice” contains only
functional information, but no primitive information. However, in general cases, each word may contain both primitive and functional information. The functional information of a word usually appears as a word class, such as part of speech. For example, in translation, a sequence of part-of-speech tags determines the output sentence structure (functional), and primitive information of each input word determines the corresponding output word.

To explore whether the proposed method works when a word contains both primitive and functional information, we introduced an extension of SCAN dataset for this purpose. We designed a task SCAN-ADJ to identify adjectives in a sentence, and place them according to their word classes. The adjectives have three word classes: color, size, and material. The word classes appear in random order in input sentences, but they should be in a fixed order in the expected outputs. For example, an input sentence is “push blue metal small cube”, and the expected output is “PUSH CUBE SMALL BLUE METAL”. For another input “push small metal blue cube”, the expected output is the same. Please see Table 9 in Appendix for more details.

To evaluate compositional generalization, similarly to the SCAN setting, the material, rubber, appears only with other fixed words in training “push small yellow rubber sphere”. However, it appears with other combinations of words at test time. To solve this compositionality problem, a model needs to capture functional information of adjectives to determine their word classes, and primitive information to determine the specific value of the word in its class.

The experiment results are shown in Table 4. The proposed method achieves 100.0% (±0.0%) accuracy, and as a comparison, a standard LSTM model can only achieves 2.1% (±2.2%) accuracy. Please refer to Appendix C for more details. This indicates that the proposed method can extend beyond simple structures in the SCAN dataset to flexibly adapt for words containing both primitive and functional information.

<table>
<thead>
<tr>
<th>Method</th>
<th>Primitive</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>82.8</td>
<td>84.3</td>
</tr>
<tr>
<td>Baseline</td>
<td>≤ 3.1</td>
<td>2.5</td>
</tr>
<tr>
<td>Proposed</td>
<td>95.0 ± 6.8</td>
<td>76.0 ± 5.5</td>
</tr>
</tbody>
</table>

Table 5: Test accuracy (mean ± std %) for few-shot learning task. The baseline methods are from Lake et al. (2019). The proposed method achieves performance superior to humans in Primitive task.

3.4 Few-shot Learning Task

We also probed the few-shot learning ability of the proposed approach using the dataset from Lake et al. (2019). The dataset is shown in Appendix Figure 4. In the few-shot learning task, human participants learned to produce abstract outputs (colored circles) from instructions in pseudowords. Some pseudowords are primitive corresponding to a single output symbol, while others are function words that process items. A primitive (“zup”) is presented only in isolation during training and evaluated with other words during test. Participants are expected to learn each function from limited number of examples, and to generalize in a compositional way.

We trained a model from all training samples and evaluate them on both Primitive test samples and all test samples. We used a similar configuration to SCAN task (Appendix D) to train the model, and compared the results with baselines of human performance and conventional sequence to sequence model (Lake et al., 2019). In the Primitive task, we used the average over sample accuracies to estimate the accuracy of human participants, and the upper bound accuracy of the sequence-to-sequence model.

Table 5 shows the results that the proposed method achieves high accuracy on both test sets. In the Primitive task, the proposed method beats human performance. This indicates that the proposed method is able to learn compositionality from only a few samples. To our best knowledge, this is the first time machines beat humans in a few-shot learning task, and it breaks the common sense that humans are better than machines in learning from few samples.

3.5 Compositionality in Machine Translation

We also investigated whether the proposed approach is applicable to other sequence-to-sequence problems. As an example, we ran a proof-of-
concept machine translation experiment. We consider the English-French translation task from Lake and Baroni (2018). To evaluate compositional generalization, for a word “dax”, the training data contains only one pattern of sentence pair (“I am daxy”, “je suis daxiste”), but test data contains other patterns. Appendix E provides more details on dataset and model configuration.

Compared to the baseline method, the proposed method increases sentence accuracy from 12.5% to 62.5% (±0.0%). Further, we find that other predicted outputs are even correct translations though they are different from references (Table 10 in Appendix). If we count them as correct, the adjusted accuracy for the proposed approach is 100.0% (±0.0%). This experiment shows that the proposed approach has promise to be applied to real-world tasks.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>12.5</td>
</tr>
<tr>
<td>Proposed</td>
<td>62.5</td>
</tr>
</tbody>
</table>

Table 6: Test accuracy (%) for machine translation task. The baseline method is from Lake and Baroni (2018).

4 Discussion

Our experiments show that the proposed approach results in significant improvements on many tasks. To better understand why the approach achieves these gains, we designed the experiments on SCAN domain to address the following questions: (1) Does the proposed model work in the expected way that humans do (i.e., visualization)? (2) What factors of the proposed model contribute to the high performance (i.e., ablation study)? (3) Does the proposed method influence other tasks?

4.1 Visualization of Attention Maps

To investigate whether the proposed method works in the expected way, we visualize the model’s attention maps. These maps indicate whether the sequence-to-sequence model produces correct position sequences. We visualized these activations for one sample from the Jump task and one from the TurnLeft task (Figure 3). In both figures, the horizontal dimension is the input sequence position, and the vertical dimension is the output position. Each figure is an $m \times n$ matrix, where $m$ and $n$ are output and input lengths, including end-of-sentence symbols. The rows correspond to $m$ attention maps, where the $i$th row corresponds to the attention map for the $i$th output ($i = 1, ..., m$). The values in each row sum to 1.

We expect that each attention map attends on the corresponding primitive input word. For the output end-of-sentence symbol, we expect that the attention is on the input end-of-sentence symbol. The visualization in both figures align with our expectation. When the model should replicate some input word multiple times, it correctly attends on that word in a manner similar to human understanding. In the Jump example, “jump” appears twice in the input. As expected, the generated attentions are on the second instance for the early part of the output, and on the first instance for the later part of the output. Similarly, the attentions are correct in the TurnLeft example with two words of “left”.

These visualizations demonstrate that the models work in the way we expected, correctly parsing the organization of complex sentences. We have successfully encoded our prior knowledge to the network, and enabled compositional generalization.

4.2 Ablation Study

We conducted an ablation study to find out what factors of the proposed approach contribute to its high performance. The main idea of the proposed approach is to use two representations and corresponding entropy regularization. Also, for autoregressive decoders, the approach does not use the last output rather than feeding it as input for decoding. Based on that, we designed the following ablation experiments.

(A) Use only one representation.
(B) Do not use primitive entropy regularization.
(C) Do not use function entropy regularization.
(D) Both B and C.
(E) Use the last output as input in decoding.

The results are summarized in Table 7. Experiment A shows that the accuracy drops in both tasks, indicating that two representations are important factor in the proposed approach. When there are two representations, we find the following from experiments B, C and D. For Jump task, the accuracy drops in these three experiments, indicating both primitive and function entropy regularization are necessary. For TurnLeft task, the accuracy drops only in D, but not in B or C, indicating at least one of entropy regularization is necessary. The dif-
Figure 3: Visualization of attention maps. The figures show that the model solves the problem in a manner similar to humans, identifying the appropriate input to map to each output position. This indicates that the generalizable prior knowledge is successfully encoded in the network.

ference between Jump and TurnLeft tasks may be because TurnLeft is relatively easier than the Jump task. Both “turn” and “left” individually appear with other words in training data, but “jump” does not. Experiment E shows that we should not use last output as next input during decoding.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Jump</th>
<th>TurnLeft</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>63.4 ± 63.3</td>
<td>55.7 ± 31.5</td>
</tr>
<tr>
<td>B</td>
<td>79.8 ± 11.3</td>
<td>99.9 ± 0.1</td>
</tr>
<tr>
<td>C</td>
<td>29.5 ± 36.3</td>
<td>99.6 ± 1.0</td>
</tr>
<tr>
<td>D</td>
<td>14.1 ± 23.2</td>
<td>64.0 ± 40.0</td>
</tr>
<tr>
<td>E</td>
<td>68.8 ± 15.0</td>
<td>50.3 ± 48.8</td>
</tr>
<tr>
<td>Proposed</td>
<td>98.8 ± 1.4</td>
<td>99.7 ± 0.4</td>
</tr>
</tbody>
</table>

Table 7: Ablation: Test accuracy (mean ± std %) for SCAN Primitive tasks.

### 4.3 Influence on Other Tasks

To find whether the proposed method affects other tasks, we conducted experiments on Simple and Length tasks in SCAN dataset. In Simple task, test data follows the same distribution as the training data. In Length task, test commands have longer action sequences than training. Simple task does not require compositional generalization, and Length task requires syntactic generalization, so that they are beyond the scope of this paper. However, we still hope to avoid big performance drop from previous methods. We used the same configuration (Appendix A) as SCAN Primitive tasks. The results in Table 8 confirms that the approach does not significantly reduce performance in other tasks.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Simple</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>L&amp;B best overall</td>
<td>99.7</td>
<td>13.8</td>
</tr>
<tr>
<td>L&amp;B best</td>
<td>99.8</td>
<td>20.8</td>
</tr>
<tr>
<td>RNN +Attn -Dep</td>
<td>100.0 ± 0.0</td>
<td>11.7 ± 3.2</td>
</tr>
<tr>
<td>GRU +Attn</td>
<td>100.0 ± 0.0</td>
<td>18.1 ± 1.1</td>
</tr>
<tr>
<td>Ptr-Bahdanau</td>
<td>13.4 ± 0.8</td>
<td></td>
</tr>
<tr>
<td>Ptr-Loung</td>
<td>16.8 ± 0.9</td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>99.9 ± 0.0</td>
<td>20.3 ± 1.1</td>
</tr>
</tbody>
</table>

Table 8: Test accuracy (mean ± std %) for other SCAN tasks. L&B is from Lake and Baroni (2018). RNN and GRU are best models for each task from Bastings et al. (2018). Ptr-Bahdanau and Ptr-Loung are from Kliegl and Xu (2018). The proposed method does not significantly reduce performance.

### 5 Related Work

Human-level compositional learning has been an important open challenge (Yang et al., 2019), although there is a long history of studying compositionality in neural networks. Classic view (Fodor and Pylyshyn, 1988; Marcus, 1998; Fodor and Lepore, 2002) considers conventional neural networks lack systematic compositionality. With the breakthroughs in sequence to sequence neural networks for NLP and other tasks, such as RNN (Sutskever et al., 2014), Attention (Xu et al., 2015), Pointer Network (Vinyals et al., 2015), and Transformer (Vaswani et al., 2017), there are more contemporary attempts to encode compositionality in sequence to sequence neural networks.

There has been exploration on compositionality in neural networks for systematic behaviour (Wong and Wang, 2007; Brakel and Frank, 2009), count-
ing ability (Rodriguez and Wiles, 1998; Weiss et al., 2018) and sensitivity to hierarchical structure (Linzen et al., 2016). Recently, many related tasks (Lake and Baroni, 2018; Loula et al., 2018a; Liška et al., 2018; Bastings et al., 2018; Lake et al., 2019) and methods (Bastings et al., 2018; Loula et al., 2018a; Kliegl and Xu, 2018; Chang et al., 2018) using a variety of RNN models and attention mechanism have been proposed. These methods make successful generalization when the difference between training and test commands are small.

Our research further enhances the capability of neural networks for compositional generalization with two representations of a sentence and entropy regularization. The proposed approach has shown promising results in various tasks of instruction learning and machine translation.

Compositionality is also important and applicable in multimodal problems, including AI complete tasks like Image Captioning (Karpathy and Fei-Fei, 2015), Visual Question Answering (Antol et al., 2015), Embodied Question Answering (Das et al., 2018). These tasks requires compositional understanding of sequences (captions or questions), where our method could help. Compositional understanding could also help enable better transfer over time that could improve continual learning (Zenke et al., 2017; Aljundi et al., 2018; Chaudhry et al., 2019) ability of AI models.

6 Conclusions

This work is a fundamental research for encoding compositionality in neural networks. Our approach leverages prior knowledge of compositionality by using two representations, and by reducing entropy in each representation. The experiments demonstrate significant improvements over the conventional methods in five NLP tasks including instruction learning and machine translation. In the SCAN domain, it boosts accuracies from 14.0% to 98.8% in Jump task, and from 92.0% to 99.7% in Turn-Left task. It also beats human performance on a few-shot learning task. To our best knowledge, this is the first time machines beat humans in a few-shot learning task. This breaks the common sense that humans have advantage over machines in learning from few samples. We hope this work opens a path to encourage machines learn quickly and generalize widely like humans do, and to make machines more helpful in various tasks.

Acknowledgments

We thank Kenneth Church, Mohamed Elhoseiny, Ka Yee Lun and others for helpful suggestions.

References


