ATNLP 2020 University of Copenhagen Final Report

Anonymous exam submission

Abstract

Transformer model architecture has been immensely popular among the NLP community since its invention. There is mounting evidence that the architecture significantly outperforms RNNs on a range of different task. In this paper, this evidence is strengthened on the SCAN task, where RNNs perform very well only for a subset of commands that are translated into corresponding actions.

1 Introduction

Language is a set of sentences constructed from finite number of elements. But combining the finite elements can produce infinitely many sentences. (Chomsky, 2002) Systematic compositionality is the ability to make use of this language property. Only humans have long been known to exhibit the systematic compositionality. Researchers believe that computational linguistics can crack the human monopoly in this area, but until now, even state-of-the-art models are failing to generalize compositionally. (Marcus, 2001) In particular, (Fodor and Pylishin, 1995) likens the process to the thought experiment where if a person knows how to "walk", "walk twice" and "jump", then the person should naturally know how to "jump twice" (Figure 1

Lake and Baroni (Lake and Baroni, 2018) have developed SCAN dataset to standardize the research in the area. The dataset has also provided a suitable benchmark for many model architectures attempting to tackle the problem of systematic compositionality. SCAN dataset contains different training and testing data splits that can assess the performance of models on several levels. In addition to that, Lake and Baroni apply popular model architectures for neural machine translation and evaluate their performance on different splits. The aim of this paper is to summarize their results, explain the process of reimplementation of the models, report the results of reimplementation and apply a popular transformer architecture (Vaswani et al., 2017) to demonstrate how it can cope with the problem of compositional systematicity.

2 SCAN task description

The SCAN dataset[1] tests the compositionality on a set of non-ambiguous commands and their corresponding actions that the model should perform. There are six possible different actions - JUMP, RUN, LOOK, WALK, LTURN, RTURN. Each has a corresponding command alongside combinatorial commands like "twice", "around", "and" and others. The goal of the model is to find an interpretation function, so that it can respond to previously never seen commands correctly (Lake and Baroni, 2018).

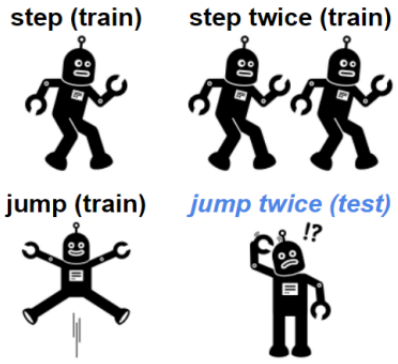

Figure 1: Illustration of machine's lack of systematicity (Li et al., 2019)

Lake and Baroni found that for some of the commands, the recurrent neural networks (RNN) perform particularly badly. This suggests that the RNNs remain dependent on multitude of data in order to be applicable, which is a symptom of lack of systematicity.

[1] https://github.com/brendenlake/SCAN

3 Experiments

Lake and Baroni run four different experiments, but since the experiment 4 is a proof-of-concept, the reimplementation replicates only the first three. The experiments are run on three types of data splits (with additional derived splits). The splits used are:

- simple 80/20 split (along with 1, 2, 4, 8, 16, 32 and 64% variations).
- split using short action sequences at training and long at testing time.
- split using only over-sampled primitive "turn left" and "jump" commands (along with splits of their composed commands).

These splits provide an opportunity to test how the generalization takes place from random previously unseen commands, from short into long action sequences and from primitive into complex commands.

4 Lake and Baroni models

Lake and Baroni implement a range of popular sequence-to-sequence (seq2seq) networks where two recurrent networks are trained to learn the mapping between input and output sequences (Sutskever et al., 2014). In more detail, seq2seq networks consist of encoder and decoder networks with the same structure (parameters and weights can be different). Encoder is used to create low-dimensional vector representation of input commands word-by-word. This representation is then passed on to the decoder to produce an output of consecutive actions to be compared to the ground truth. Once the loss is calculated based on the difference between the predictions and ground truth, backpropagation algorithm takes care of updating the weights.

The RNNs used are simple recurrent newtworks (SRNs; (Elman, 1990), long short-term memory (LSTMs; (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs; (Chung et al., 2014). In the recent years, attentional mechanism (Bahdanau et al., 2014) proved to be effective in similar applications. Thus it has been included in the search for the best model. Through the hyperparameter search the overall-best model was found to be 2-layer, 200 hidden units LSTM without attention and 0.5 dropout.

Finally, the models were trained using 100,000 iterations over the command-action pair. The optimization algorithm used was Adaptive Moment Estimation (ADAM) using a learning rate of 0.001. All gradients with norm larger than 5 were clipped. There was also 50% chance that the ground truths were passed on to the next iteration instead of the produced outputs - a technique called teacher forcing (Williams and Zipser, 1989).

Apart from the best-overall model, Lake and Baroni report the best model for each of the experiments. The summary of the results from these experiments can be seen in table 1.

5 Reimplementation

On of the two contributions of this paper is to reimplement the results of Lake and Baroni and report the results. Similar to Lake and Baroni, the implementation took place using PyTorch package in Python. The implementation is based on PyTorch tutorial on seq2seq network with attention.[2] The main steps of the reimplementation follow. First, we mapped each preprocessed command and action in separate dictionaries, that were loaded in the torch.utils.data.Dataset class. We used the instances of this class for our data loader when pulling in the itarations during the training. Encoder from the tutorial has been adjusted to accommodate a switch between LSTMs and GRUs. The output of the encoder is the hidden layer, which initialization is implemented as a method in encoder class and the output. Encoder outputs and hidden layer, together with the input are then passed in the decoder. If attention is used, implementation of Bahdanau (Bahdanau et al., 2014) follows within the forward method of decoder class. The hidden layer, output (prediction) and attention weights are the output from the decoder.

The training loop follows, which ensures that input and target tensor enter the network and Cross Entropy Loss is calculated to recalculate the weights through backpropagation. The training is repeated 100,000 times, always taking the following training pair in the training data loader. Thus in case of 20,000 training instances, the network runs for five epochs. The evaluation function follows, which replicates the training function, but

[2] https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Experiment	Best model	Best result	Our Best result	Best overall	Our best overall
Exp 1	2-layer LSTM, 200 hid. units, no attention, no dropout	99.8%	99.84%	99.7%	99.7%
Exp 2	1-layer GRU, 50 hid. units attention, 0.5 dropout	20.8%	21.48%	13.8%	13.05%
Exp 2 with Oracle	1-layer GRU, 50 hid. units attention, 0.5 dropout	60,2%	55.79%	23.6%	-
Exp 3 LTURN	1-layer GRU, 100 hid. units attention, 0.1 dropout	90.3%	82.65%	90.0%	90.12%
Exp 3 JUMP	1-layer LSTM, 100 hid. units attention, 0.1 dropout	1.2%	0.02%	0.08%	0%

Table 1: Results of Lake and Baroni experiments and our reimplementation of them (Lake and Baroni, 2018).

without updating the gradients. The results from the implementation are displayed in the table 1 together with the best results from Lake and Baroni implementation.

6 Discussion of reimplementation results

In overall, the reimplementation was considered a success. Most of the experiments achieved comparable results to Lake and Baroni. However, especially in models using attention mechanism, the results obtained displayed much higher variance. While we should expect some variance, due to the models being non-deterministic, particularly in experiment 3 testing composed commands the variance is prominent (see 2). Since this problem does not occur in no-attention models, it is expected that the attention implementation has been the source of the issue. Due to that, attention visualization is used to examine the connections between commands and actions that get triggered the most when 'translating' between the sequences (see 3).

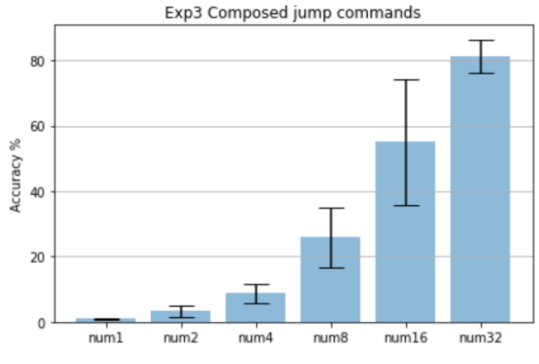

Figure 2: In experiment 3 with composed commands, the variance is especially prominent.

It can be observed, that higher emphasis is laid

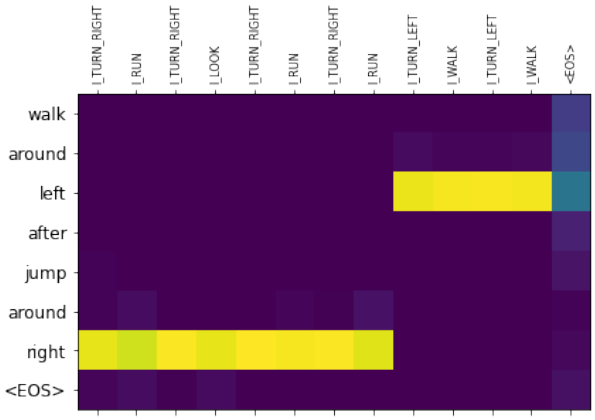

Figure 3: The attention weights connecting commands and actions are 'focusing' on single commands rather than switching between them. The chart shows 2 strong attention lines that are only able to correctly predict the sequence of actions. This pattern is observed in most composed instances and is reversed if used "and" command (the top attention line would be on the left)

on the commands repeating actions rather than the actions. Due to that, the model seems to be guessing between the reduced number of options and thus often ends up making incorrect decisions. This guessing may also be the cause of such high variance, due to randomness in being more correct in some cases.

7 Transformer

Since the most of the poor performing tasks had been selection models using attention in Lake and Baroni's model architecture grid search. Thus, a more advanced attention mechanism could potentially improve the results. This idea motivated the implementation of a Transformer architecture (Vaswani et al., 2017) on the problem. The architecture removes the recurrence from the model and

focuses on attention only.

It consists of N encoder and decoder layers. Each encoder layer has two sub-layers consisting of self-attention and position-wise feed-forward network. Layer residuals are included in the next layer and normalized. Each decoder layer consists of the same sub-layers as the encoder, but also performs multi-head attention over the encoder's outputs. There is also one change in self attention sub-layer, where the subsequent positions are masked in order to prevent them from being involved in the predictions making.

The type of attention applied in Transformer is called Scaled Dot-Product Attention. It consists of queries, keys and values matrices with equal dimensions and combined in the formula:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

Multi-head element of attention allows to represent the information in different subspaces. With single attention they are simply averaged and therefore prevent from retaining the information. The outputs of the attention are then concatenated in the following way:

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned} \quad (2)$$

Even though we calculate several heads of the attention, the computational cost is not that significant, since for each head we use reduced dimensionality of d_{model}/h.

Another sublayer that the encoder and decoder layers contain is position-wise feed-forward network. This consists of two linear transformations activated by ReLU in between. See:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (3)$$

The parameters across different layers are not shared, even though they are performed the same way. In (Vaswani et al., 2017) they use 4-times higher dimensionality of the network compared to the hidden layers.

Lastly, since the Transformer does not follow recurrence, the order of the sequence is recorded using positional encoding with the same dimensions as hidden layer, so they can be summed.

8 Transformer implementation

The transformer for the SCAN task has been implemented using different parameters than the one presented by Vaswani et al. The idea behind is to use the same number of layers, dimensions of hidden layer as with the LSTMs and GRUs. In the case of additional parameters, 4 attention heads and dimensions of position-wise feed-forward network $4 * d_{\text{model}} = 800$.

The implementation of Ben Trevett[3] was followed in combination with HarvardNLP[4] as Harvard's has by purely personal opinion clearer encoder and decoder modules. Training and evaluation parts were implemented in a similar fashion as LSTMs and GRUs with slight difference of including masking in the evaluation section, according to jlrussin on GitHub.[5] Hyperparameters like learning rate, clipping we set to the same values as Lake and Baroni's implementation.

9 Transformer results and discussion

The results that came out of the transformer architecture were overwhelmingly good. On all tasks (and their variations) and subsets it has achieved 100% with exception of primitive jump train/test split where it achieved 99.7% as five-run average. It is also notable that unlike the RNN architecture, transformer was trained only on 5,000 iterations. Since the model is able to achieve such high results in so few iterations, it would be redundant to run 100,000 of them. The time of each iteration is comparable to the one of RNNs, thus the time transformer requires for the training is around 20-times shorter.

Since the hypothesis of why transformer architecture should perform better on the task than the RNNs was due to more advanced attention mechanism. The investigation of why the results turned out to be so good will be verified based on attention matrices visualisation. In the encoder attention heads, we can clearly see that the weights are differently catered in each head. This potentially improves the decision making when predicting the

[3] https://github.com/bentrevett/pytorch-seq2seq/blob/master/6%20-%20Attention%20is%20All%20You%20Need.ipynb

[4] http://nlp.seas.harvard.edu/2018/04/03/attention.html

[5] https://github.com/jlrussin/transformer_scan/blob/master/test.py

correct action sequences. The attention weights from all four heads can be seen in 7, the two that confirm the quality of the model are pictured in figures 4 and 5. As seen in appendix, the other 2 heads do not convey that much information, they do however increase the performance of the model. Note that the attention plots the same command-action sequence we presented in RNN part for comparison.

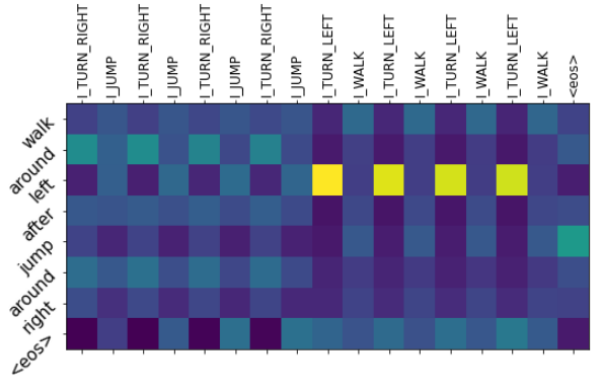

Figure 4: In the head we can see that it caters for the command-action left-I_TURN_LEFT as well as slightly for walk-I_WALK and around for the first half of the action execution.

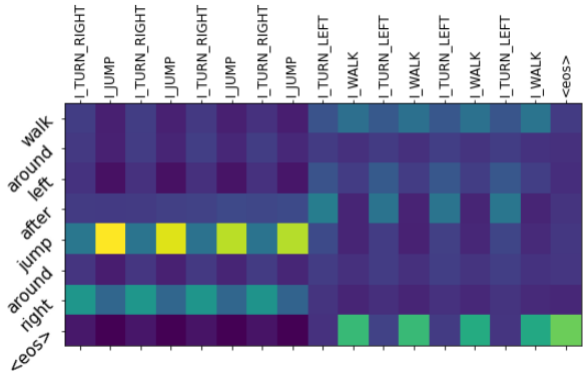

Figure 5: Head 4 caters for the jump-I_JUMP pair, as well as right turn. Small attention is also given to the word after for actions in the second half of the sequence, presumably covering the order. Notable attention increase takes place at combination of <EOS> tag, implying that it comes after the I_WALK action.

To further verify the validity of the performance of the model, evaluating different literature analysing the SCAN task was considered. There are not that many papers involving combination of SCAN task and transformer architecture, however. (Furrer et al., 2020) is basing his research on different set of tasks in which vanilla transformers performance is mixed. If, however, transformers do not perform equally well on all task then the results of implementation performs contradictorily good. Due to that, one more implementation of transformer[6] has been tested. This implementation also showed much improved results compared to the Lake and Baroni's implementation, but it was lower for for the "primitive jump" task from experiment 3, where it achieved accuracies ranging from 47-71%.

The main difference between the two implementation was masking. While jlrussin's implementation used to torch module for multi-head attention, our implementation used the module from Ben Trevett's tutorial. The torch module takes in and outputs the key padding mask and the attention mask separately, while in the Ben Trevett's tutorial they are combined. If masking has failed in the implementation, then it would be possible that overflowing ground truth influences the predictions significantly. However, with low number of iterations the models tends to predict only sequences consisting of N-number of the same actions, thus it seems reasonable to argue that if the ground truth somehow influence the predictions, then at least a few of them would be correct. In such cases, however, the probabilities are around 0%. This behaviour slightly resembles systematic compositionality, since in case of humans it would be reasonable to think that if the generalization is possible, then it is possible absolutely.

10 Conclusion and Future Work

While the last part of the paper was bold enough to suggest the systematic compositionality of the presented model, it needs to be acknowledged that it could be unlikely. Thus, the future research should look further into what is masked in the model. If the masking is implemented in a correct manner, then where is the cut-off for the model to achieve near or 100% performance.

In any case, this paper demonstrates that the transformer architecture indeed handles the SCAN tasks with much better results than traditional RNNs. The paper can thus be simply considered to be adding to the evidence over superiority of this architecture over range of different tasks.

[6] https://github.com/jlrussin/transformer_scan

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Noam Chomsky. 2002. *Syntactic structures*. Walter de Gruyter.

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

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

J Fodor and Z Pylishin. 1995. Connectionism and cognitive structure: a critical review. *Language and intelligence.–Moscow: Progress*, pages 230–314.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*.

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

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR.

Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. 2019. Compositional generalization for primitive substitutions. *arXiv preprint arXiv:1910.02612*.

Gary Marcus. 2001. The algebraic mind.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27:3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

A Appendices

A.1 Scaled dot product and multi-head attention schemas

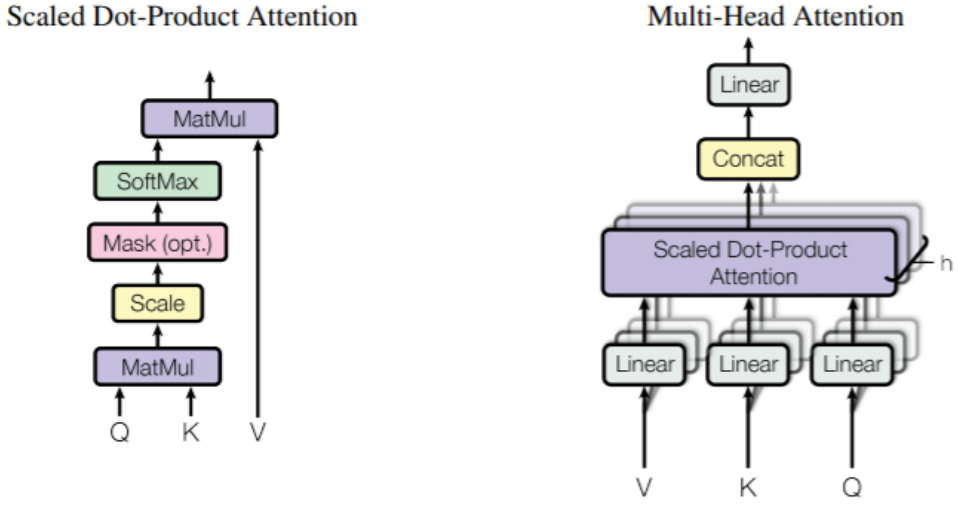

Figure 6: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel (Vaswani et al., 2017)

A.2 All four attention heads weights illustration

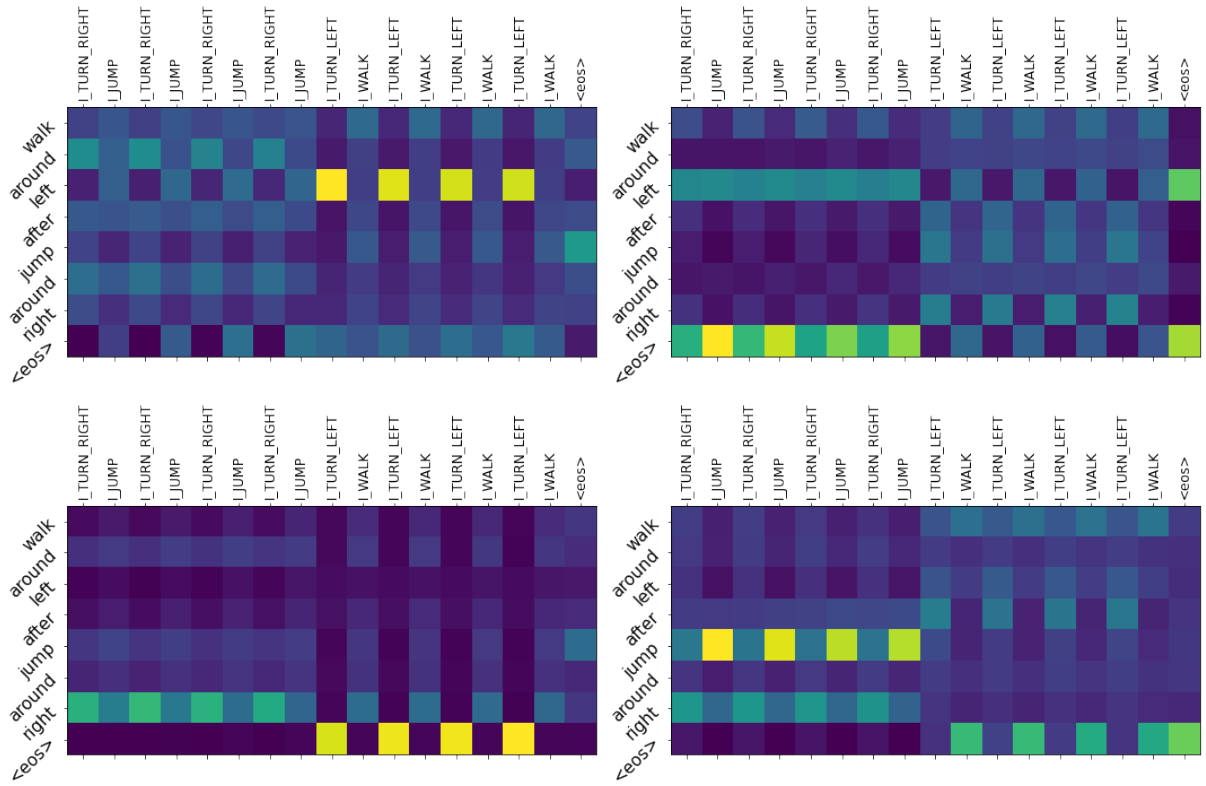

Figure 7: All 4 attention heads of the command-action pair discussed in the paper