Advanced Topics in Natural Language Processing: Individual Project

André Oskar Andersen (wpr684)
wpr684@alumni.ku.dk

1 Introduction

The following paper is my individual project for the course *Advanced Topics in Natural Language Processing* taught at the University of Copenhagen. The aim of the paper is to cover my (group's) attempt at replicating Lake and Baroni's results from their paper *Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks*. Further, the paper aims to cover my attempt at repeating some of the experiments from Lake and Baron, but instead with a transformer-based model (Lake and Baroni, 2017).

2 The SCAN Tasks

For the data, the SCAN dataset will be used (Lake and Baroni, 2017). The goal with this dataset is for a model to learn in a supervised manner, how to translate commands into a sequence of actions. The GitHub repository of the SCAN dataset has already split the dataset into various data-splits, such that one can easily access the proper data for a given experiment [1].

3 Implementing the Data Loader

Our very first task is to implement a dataloader for the SCAN dataset. We did this in a very configurable manner, such that for each experiment the corresponding data can easily be provided. This was done by providing a `split`-parameter, that decides what data-split to use, and a `split_variation`-parameter, that decides on the train/test-split ratio.

Further, we implemented a seperate class for the vocabulary that maps an input token to an unique integer, such that one can convert between tokens and their respective integer-representations. Further, this class is provided with constants for four tokens that represent start-of-sentence, end-of-sentence, out-of-vocabulary, and a padding token, such that no other token is mapped to the integer-representations of these four tokens.

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

4 Models

For the models we will be implementing various seq2seq-models, some of which are experiment specific and will be covered in their respective experiment-part. The remaining models will be covered in Subsection 4.1 and 4.2. Further, some of these models will be making use of attention as described by Lake and Baroni (Lake and Baroni, 2017).

4.1 The Overall-best Model

The overall-best model was found by a hyperparameter search. The architecture consists of a 2-layer LSTM with 200 hidden units per layer, no attention, and dropout with a dropout-ratio of 0.5 (Lake and Baroni, 2017).

4.2 The Transformer Model

The transformer model was implemented from scratch by using the Transformer-class from PyTorch [2], which is then followed by linear layer. Instead of training the model from completely scratch, we could have chosen to use a pre-trained model in a fine-tuning or in-context learning setup, however, we chose not to do so, as we expected to gain a better understanding of transformers by implementing one from scratch.

When implementing the transformer we aim at following the setup introduced by Vaswani *et al.* as closely as possible. Thus, the input and output is embedded by using a learned embedding, which we den multiply with the square root of the embedding size and use for computing the positional

[2] https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html

embedding in a similar maner as Vaswani *et al.* (Vaswani et al., 2017). Due to memory limitations, we did however decide to reduce the number of parameters in the model. Thus, the embedding and final linear layer both have a dimensionality of 200. Further, the transformer uses 8 heads, 2 encoder and decoder layers and no dropout, which was found by following a small grid-search on the dataset for the first task of experiment 1.

4.3 Training Details

All models was trained on $100,000$ randomly chosen samples using the ADAM opimizer with the default PyTorch settings and a learning rate of 0.001, as well as clipping the gradients with a norm larger than 5. For the transformer-based models, the learning rate was reduced with a factor of 0.1 for every 500 iteration. The non-transformer models used a batch-size of 1, whereas the transformer-based models used a batch-size of 64. Finally, for the non-transformer models, at each training iteration a bernoulli random variable decided whether to use teacher forcing or not (Williams and Zipser, 1989). All presented results are based on five runs to increase the confidence of the results.

5 Experiment 1

The first experiment consists of two tasks. For the first task, the dataset was randomly split into a training set, consisting of 80% of the data, and a test set, consisting of the remaining 20%. The training set provides a wide coverage of the task space, such that the goal of the test set is to evaluate whether the model has learned to generalize by recombining pieces of the training commands to interpret the new ones in the test set. For the second task, a model is retrained on various training/test-splits.

The experiment-best model found by Lake and Baroni (Lake and Baroni, 2017) was an LSTM with no attention, 2 layers of 200 hidden units and no dropout.

5.1 Results

For the first task, Lake and Baroni achieved an accuracy of 99.7% and 99.8% for the overall-best model and the experiment-best model, respectively (Lake and Baroni, 2017). We achieved an accuracy of 99.2% and 99.8% for the overall-best model and the experiment-best model, respectively, leading to us believing, that we have correctly implemented the two models. The difference in performance between the two overall-best models is possibly due to the randomness in the initialization of the models, as this difference in the performance is so small.

For the second task, we have in Figure 1 visualized the expected outcome, as well as the results we have obtained with our implementations. Looking at the figure we can see, that our implementation of the overall-best model achieves results that are very similar to the results reported by Lake and Baroni (Lake and Baroni, 2017), making us further believe, that we have implemented the model correctly. On the other hand, if we compare the results of our transformer model to the two other plots, we can see, that the transformer model yields much worse results, than the two other models. We see, that when using just 8% of the data for training, the LSTM models delivers the same results as when the transformer model uses 64% of the data.

6 Experiment 2

The second experiment consists of one task and two additional analyses. In the task, the models must bootstrap to commands requiring longer action sequences than those seen in training. For this, we use all $16,990$ commands that correspond to an action sequence of 22 actions, for training the model, whereas the test set consists of the remaining $3,920$ commands that corresponds to actions sequences of lengths from 24 to 48 actions.

For the first additional analysis, Lake and Baroni examines the greedy decoder for search-related errors. This was done by confirming for almost every error, that the network prefers it self-genereated output sequence to the target output sequence. For the second additional analysis, Lake and Baroni studied whether the difficulty with long sequences would lessen if the number of action tokens were provided by an oracle at evaluation time (Lake and Baroni, 2017).

The experiment-best model found by Lake and Baroni was a GRU with attention, one hidden layer with 50 units, and dropout with a dropout-ratio of 0.5 (Lake and Baroni, 2017).

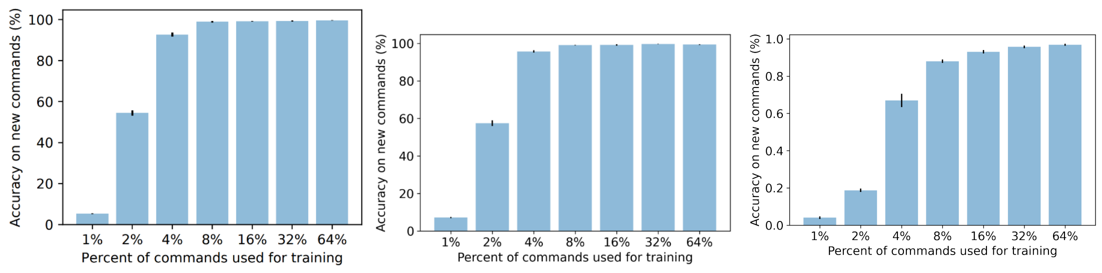

Figure 1: Results on the second task of the first experiment. Left: Reported results of Lake and Baroni (Lake and Baroni, 2017). Middle: Our obtained results, using the overall-best model. Right: Our obtained results, using the transformer model.

6.1 Results

For the task, Lake and Baroni reported an accuracy of 20.8% and 13.8% for the experiment-best model and overall-best model, respectively (Lake and Baroni, 2017). We initially achieved an accuracy of 9% and 13% for the experiment-best model and overall-best model, respectively, however, by incorporating some feedback we received at the midway-evaluation, we have successfully increased the performance of the experiment-best model to 19%. If we compare this to the results of our transformer model, which obtain an accuracy of 7.6%, we see, that the recurrent neural networks completely outperforms the transformer model.

Figure 2 goes more in depth with the results of the three developed models. Similarly to the reported results of Lake and Baroni (Lake and Baroni, 2017), our overall-best model also only performs decently on the shortest action sequences, with very similar results for action sequences with a length of 24 or 25 actions. On the other hand, our model does not perform as well as Lake and Baroni's models on action sequences with a length of 28 or 32 actions.

Also for the transformer model, the best performing samples have an action sequence length of 24 or 25 actions. The model does, however, not perform as well as the two overall-best models. On the other hand, the transformer model actually outperforms the two overall-best models on longer action sequences. We theorize, that this is due to the fact, that vanishing and exploding gradient has a much smaller effect in transformers than it has in GRUs, making the transformer perform better on longer sequences, as the transformer at the end of the sequence is better at "remembering" information from early in the sequence.

As with Lake and Baroni, the bottom panel of Figure 2 shows, that both of the overall-best models only performs well on commands with 8 or 9 tokens. As stated by Lake and Baroni, this is because, the longest action sequences from the training set corresponds to commands of 8 or 9 tokens, hence why the model learns to correctly generalize only in those cases. Further, our overall-best model performs very similarly to the model by Lake and Baroni (Lake and Baroni, 2017). The transformer model follows a similar pattern, but with a lower accuracy.

Based on these results we believe, our overall-best model has been implemented correctly. This is because, we achieve very similar results to the ones reported by Lake and Baroni (Lake and Baroni, 2017). The only major differences are in the performance on action sequences with a length of 28 or 32 actions, however, we believe that this is due to randomness in the initialization of the networks.

For the first test, Lake and Baroni just report, that for almost every error, the network preferred its self-generated output sequence to the target output sequence (Lake and Baroni, 2017). When we perform a similar test on our models, including the transformer-based models, we also get, that for the majority of the time our models prefer their self-generated output sequences to the target output sequences.

For the second test, Lake and Baroni increased the performance of the overall-best and experiment-best models to 23.6% and 60.2%, respectively. When we perform a similar test, we achieve an accuracy of 7.7% and 12.8% for the overall-best

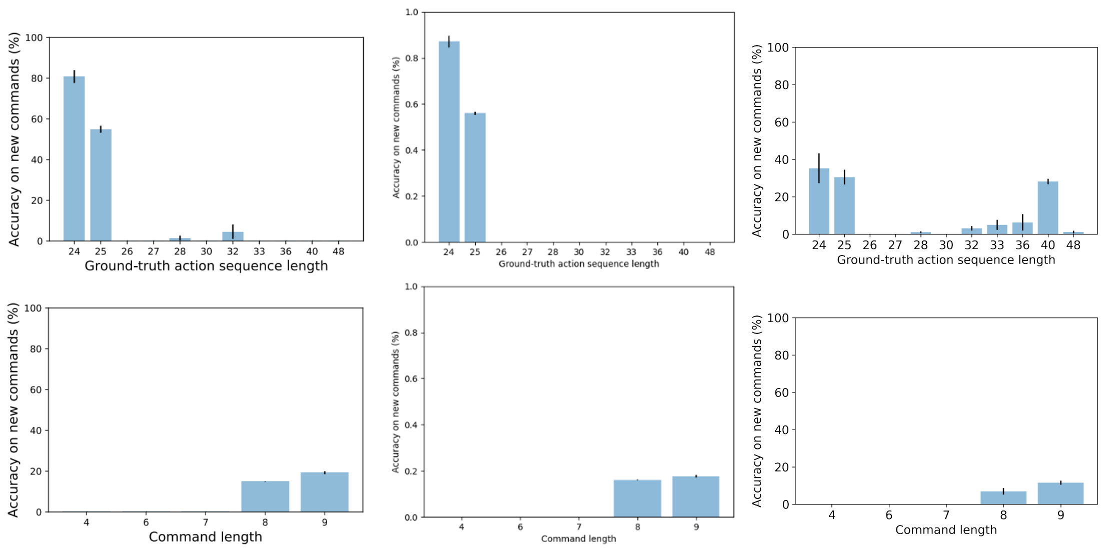

Figure 2: Zero-shot generalization to commands with action sequence lengths not seen in training. Top: accuracy distribution by action sequence length. Bottom: accuracy distribution by command length. Left: Reported results of the overall-best model by Lake and Baroni (Lake and Baroni, 2017). Middle: Results of our overall-best model. Right: Results of our transformer model.

and experiment-best models, respectively. In this case, we suspect, that our implementation is incorrect, since these accuracies are lower than when we are not using the oracle, but due to the time limit, we unfortunately did not have the time to correct this implementation. When performing a similar test on the transformer-based model, our accuracy increases to 43.4%, outperforming Lake and Baroni's overall-best model (Lake and Baroni, 2017).

7 Experiment 3

For the third experiment the models are trained on the primitive commands only denoting a basic action, as well as all primitive and composed commands for all other actions. Then, when evaluating the model, it has to execute all composed commands for the action that it only saw in the primitive context. Two variants of the experiment based on generalizing from the "turn left"-command and "jump"-command were ran (Lake and Baroni, 2017).

For the "turn left" case, the best performing model was achieved by a GRU with one 100-dimensional layer with attention and a dropout of 0.1. For the "jump"-case, the best performing model was an LSTM with one 100-dimensional layer with attetion and dropout at 0.1 (Lake and Baroni, 2017).

7.1 Results

For the "turn left"-case, Lake and Baroni reported an accuracy of 90.3% and 90.0% for the experiment-best and overall-best model, respectively. For the "jump"-case, they report an accuracy of 1.2% and 0.08%, for the experiment-best and overall-best model, respectively (Lake and Baroni, 2017). When we tried to replicate these results, we only achieved an accuracy of 89% and 76% for the "turn left"-case for the overall-best and experiment-best models, respectively. Similarly, our models did not learn to generalize at all for the "jump"-case, as both the overall-best and experiment-best models achieved an accuracy of 0%. These results are generally so far away from the reported results, that we must have made an incorrect implementation.

Lake and Baroni further reports the 5 nearest neighbours for a samples of commands. Table 1 reports our cosine similarity on the same commands. Based on the low accuracies of our models, we would expect our cosine similarities to be smaller than the ones reported by Lake and Baroni, which is also what we experience, further confirming our belief, that we have made an incorrect implementation.

Lastly, Lake and Baroni tested the performance of the model when it is only exposed to

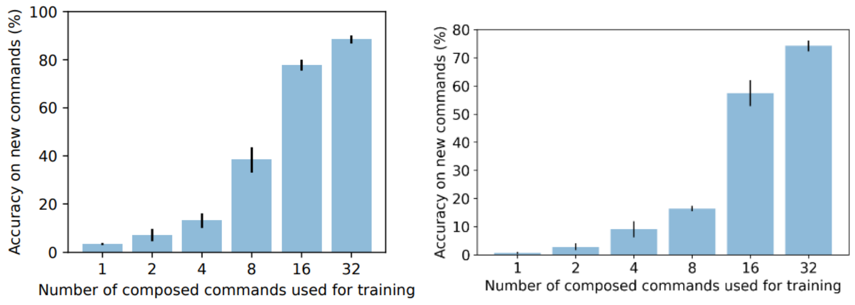

Figure 3: Zero-shot generalization after adding the primitive "jump" and some compositional "jump" commands. Left: reported results by Lake and Baroni (Lake and Baroni, 2017). Right: our obtained results.

the "jump"-command in isolation and asked to bootstrap to its compositional paradigm based on the behavior of other primitive commands. Figure 3 reports Lake and Baroni's results, as well as our attempt at replicating their results. Generally, our model follows the same pattern as Lake and Baroni's implementation, however, our implementation is heavily underperforming, as expected due to our hypothesis that we have an incorrect implementation.

8 Discussion

In the following we will be discussing our results, some future work, as well as our obtained knowledge.

8.1 Transformer Results

Through the first two experiments we have seen, how the recurrent neural networks consistently outperforms the transformer-based models. We believe this performance difference is due to the low training resource. Hassani *et al.* states, that the transformer models are very data hungry and thus requires a lot of training data (Hassani et al., 2021). Since our dataset consists of very few samples, the model cannot reach its full potential.

8.2 Future Work

If we were to work further with this project, it would be interesting to test whether using a pre-trained transformer-based model would outperform the recurrent neural networks. Hendrycks *et al.* has shown, that unsupervised pretraining in transformers are beneficial for out-of-distribution generalization (Hendrycks et al., 2020), making us believe, that using a pre-trained transformer model would yield better results.

8.3 Obtained Knowledge

Through the completion of the experiments of this paper, we have learned how to implement seq2seq-models and transformer-based models from scratch. Further, we have learned about the advantages and disadvantages of these models.

9 Conclusion

Throughout this paper we have successfully replicated the results of the first experiment and most of the second experiment of Lake and Baroni. We have argued, that training a transformer-based model from scratch cannot outperform a recurrent neural network on these tasks, as well as theorized how using a pretrained model could outperform these recurrent neural network. Further, we have attempted at replicating Lake and Baroni's results of the third experiment, however, without success (Lake and Baroni, 2017).

References

Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. 2021. Escaping the big data paradigm with compact transformers.

Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. 2020. Pretrained transformers improve out-of-distribution robustness.

Brenden M. Lake and Marco Baroni. 2017. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

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

run	jump	run twice	jump twice
look 0.09	run 0.39	look twice 0.5	walk and walk 0.24
walk 0.14	walk 0.30	run twice and look opposite right thrice -0.01	run and walk 0.27
walk after run 0.56	turn right 0.1	run twice and run right twice 0.56	walk opposite right and walk 0.08
run thrice after run 0.51	look right twice after walk twice -0.06	run twice and look opposite right twice 0.4	look right and walk 0.03
run twice after run 0.54	turn right after turn right 0.0	walk twice and run twice 0.59	walk right and walk 0.6

Table 1: Lake and Baroni's five nearest training commands for representative commands and our respective obtained cosine similarities (Lake and Baroni, 2017).