Formal Logical Reasoning With Transformers and Their Place on the Chomsky Hierarchy

Stefan Reifberger[1]

[1]Ludwig-Maximilians-Universität München and Technical University of Munich, Germany

Abstract

Building on recent interest in the capability of the transformer architecture to learn symbolic reasoning I present results in training deep learning architectures on a classical natural deduction calculus in propositional logic. For tests on novel derivations within the length of training derivations, the transformer outperforms encoder-decoder LSTMs significantly. This is despite contrary previous results for test derivations longer than seen during training.

Keywords

deep learning, logic, symbolic reasoning, transformer, Chomsky hierarchy, formal grammars

1. Introduction

Users want deep learning models like GPT to tell them the truth, not just empirical truths like Munich being the capital of Bavaria. Rather they expect them to make logical inferences of the form "If A then B" and "Not B" then "Not A" deterministically from provided information. However, deep learning models would have to emulate this deductive process via probabilistic induction: learning the most probable next words or characters given some inference rule. The crucial question now is how well they can do it and if so, why.

To make this inquiry I test an LSTM encoder-decoder and transformer trained on making derivations in a classical propositional logic. To accomplish the task, the architecture of choice must be at least context-free on the Chomsky hierarchy as the language of propositional logic is context-free see (for an explanation, see [1]). First, in Section 2 I will introduce the logic and then elaborate on recent results in Section 3. On this basis, I introduce my experimental setup and its results in Sections 4 and 5 and discuss them in Section 6.

2. The Logic

For my inquiry, I consider propositional logic. There, propositions such as the content of 'If Bob is in Munich, then Bob is in Bavaria' are represented in a symbolic language as, for example, '$(p \to q)$'. For example, by applying deductive rules, it can be derived that 'If Bob is not in Bavaria, then Bob is not in Munich,' as '$(\neg q \to \neg p)$'. More formally, this paper refers to a language $\mathcal{L}$:

Definition 1 (Alphabet of $\mathcal{L}$). *The alphabet of $\mathcal{L}$ is:*

1. *logical connectives:* $\neg, \wedge, \vee, \to$,
2. *propositional variables:* $\{p_1, \ldots, p_n\} = \mathcal{V}$,
3. *auxilliary signs:* $), ($.

Definition 2 ($\mathcal{L}$-formulas). *The set of $\mathcal{L}$-formulas, are defined by:*

1. *Each propositional variable $p_i \in \mathcal{V}$ is an $\mathcal{L}$-formula.*

OVERLAY 2024, 6th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, November 28–29, 2024, Bolzano, Italy

stefan.reifberger@tum.de (S. Reifberger)

0000-0001-8788-9591 (S. Reifberger)

2. *If A is an $\mathcal{L}$-formula, then $\neg A$ is an $\mathcal{L}$-formula.*
3. *If A and B are $\mathcal{L}$-formulas, then $(A \wedge B)$ is an $\mathcal{L}$-formula.*
4. *If A and B are $\mathcal{L}$-formulas, then $(A \vee B)$ is an $\mathcal{L}$-formula.*
5. *If A and B are $\mathcal{L}$-formulas, then $(A \rightarrow B)$ is an $\mathcal{L}$-formula.*
6. *If A and B are $\mathcal{L}$-formulas, then $(A \leftrightarrow B)$ is an $\mathcal{L}$-formula.*
7. *Nothing else is an $\mathcal{L}$-formula.*

Further, there are the following inference rules, where '$A, B, C, ...$' represent $\mathcal{L}$-formulas, Greek letters at least one $\mathcal{L}$-formula and '$\vdash$' that formulas to the right of it can be derived from the left ones:

Definition 3 (Classical logic. CL). *Classical Logic (***CL***) is the set of rules of inference:*

1. $(\bot E) \quad \bot \vdash A$
2. $(\neg E) \quad A, \neg A \vdash \bot$
3. $(\neg I) \quad$ If $\Gamma, A \vdash \bot$, then $\Gamma \vdash \neg A$
4. $(\wedge I) \quad A, B \vdash A \wedge B$
5. $(\wedge E) \quad A \wedge B \vdash A$ *and* $A \wedge B \vdash B$
6. $(\rightarrow E) \quad A, A \rightarrow B \vdash B$
7. $(\rightarrow I) \quad$ If $\Gamma, A \vdash B$ then $\Gamma \vdash A \rightarrow B$
8. $(\vee I) \quad A \vdash A \vee B$ and $B \vdash A \vee B$
9. $(\vee E) \quad$ If $\Gamma, A \vdash C$ and $D, B \vdash C,$ then $\Gamma, D, A \vee B \vdash C$
10. $(DN) \quad \neg\neg A \vdash A$

3. Transformers on the Chomsky Hierarchy

Some previous results show that simple recurrent neural networks and transformers are Turing complete [2, 3, 4, 5], which is to say that they are Turing machines and should, therefore, be capable of producing any of the grammars on the Chomsky hierarchy. The practical relevancy of the theoretical results is, however, to be questioned, as shown in a recent paper by researchers from Google Deep Mind [6]. The authors used synthetic data, where production rules of a level on the hierarchy were necessary and sufficient to produce correct expressions. These were sets of sequence-to-sequence tasks that could only be solved by automata at a minimum level on the hierarchy. During testing, the output sequences had to be longer than seen during training. Via these, they tried to test different architectures experimentally on which type of automaton they correspond to. They show that RNNs and LSTMs cannot produce languages higher on the hierarchy except when augmented with a stack or stash memory. Transformers cannot even recognize certain regular languages.

4. Experimental Setup

The plan is to generate a set of premises and conclusions in $\mathcal{L}$, where the conclusions follow logically from the premises by **CL**.[1] Different architectures are then trained on these premise-conclusion pairs. After each training epoch, model accuracy and loss are then evaluated on premise-conclusion pairs not seen during training, such as the simple: '$p, q, (p \wedge q)$'. 'p' and 'q' are the premises and '$(p \wedge q)$' is the conclusion.

Because the exact derivations in the test data have not been seen during training, success in learning these would imply that the underlying derivation rule that generated them was learned. In other words: If from '$p \wedge q, q \rightarrow p, q, p$' as ground truth in the training for training input '$p \wedge q, q \rightarrow p, p$' it had learned to produce '$r \wedge s, s \rightarrow t, s, t$' from '$r \wedge s, s \rightarrow t, t$', then I reason that it had learned the underlying rules $\wedge E$ and $\rightarrow E$.

[1] The Github repository with the implementation and produced sample formulas analyzed in Section 5.2 can be found here under https://github.com/stereifberger/master-s-thesis-finished.

All computed paths in the training and test dataset have a length of one or two derived formulas. In this output space, there are for example the following derivations that correspond to the above premise conclusion pair: '$p, q, (p \wedge q)$' and '$p, q, \mathrm{A}, (p \wedge q)$', where 'A' denotes all formulas that can be derived from 'p' and 'q' via rule application. Therefore, there may be several correct derivations for each premise-conclusion pair as in supervised learning models are typically trained on single ground truth data entries.

Therefore, I introduce a custom loss calculation. Model outputs are compared to all correct derivations. Then, Euclidean distance is used to find the closest correct derivation to the given model output.

$$Distance(x, y) = \sqrt{(x - y)^2}$$

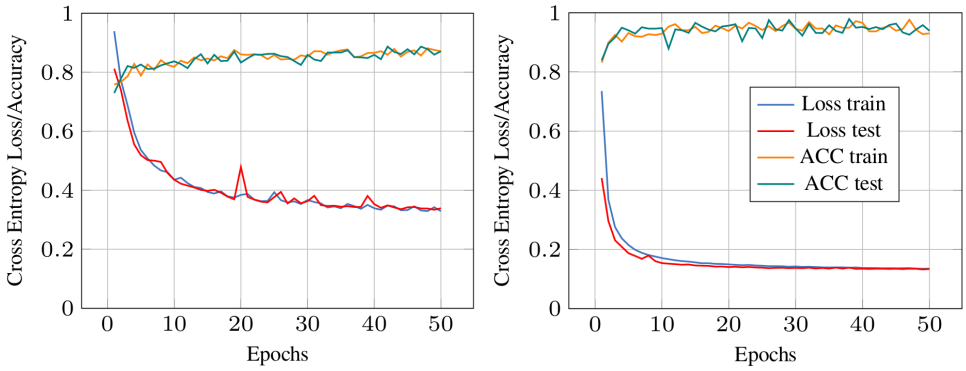
This closest derivation is taken as ground truth. With it, the cross entropy loss is calculated, which is standard for the sequence-to-sequence tasks. For the models, I consider three encoder-decoder architectures. As a baseline, a feedforward-encoder-decoder with no hidden layers is tested. As competitive architectures, LSTM encoder-decoders and the transformer architecture are tested. I test them under different architectural specifications for the number of hidden layers in both the encoder and decoder.

5. Results

The results of the two competitive models are rather different from the feed-forward network baseline.[2] This shows both in their learning curves under varied architectural specifications and the logical derivations they could produce.

5.1. Train and Test Results

The LSTM-encoder-decoder shows a learning curve (Figure 1a) with both the training and test loss consistently going down and a mean of .340 between epochs 40 and 50. Training and test accuracy steadily increased, reaching a mean of .870 accuracy at epoch 50. The transformer improves on this even further. First, as Figure 1b shows, it learns much faster than the LSTM, reaching training and test accuracy of .90 after only three epochs.



(a) LSTM-Encoder-Decoder. (b) Transformer.

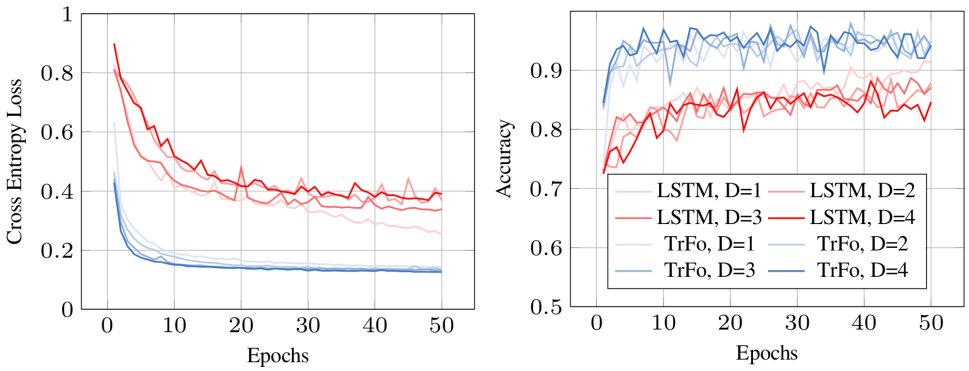
Figure 1: Results for the competitive models. The models always have one layer in the encoder and three in the decoder. The transformer's encoder has one FFN layer and five attention heads; the transformer's decoder has three FFN layers and three attention heads. Batch-size = 64.

Next the architectures' robustness towards architectural changes is tested by first testing for different depths in the decoder and then the encoder. Figure 2 shows the test loss and test accuracy with LSTMs

[2] All results are documented in https://github.com/stereifberger/master-s-thesis-finished/blob/new-cleaned-branch/results.pdf.

and transformer models with the encoder layer fixed to one and the decoder layers varied between one and four.

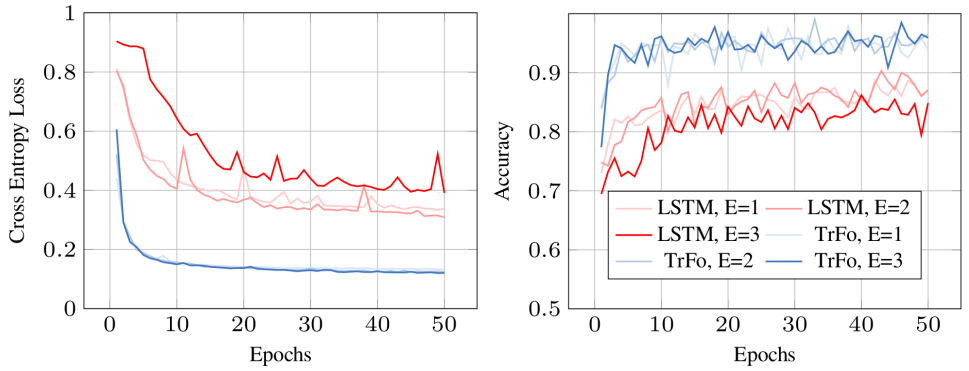
Only for the LSTM, there is some significant difference by increasing the number of decoder layers, making the performance worse. For accuracy, there is no significant difference. Because the latter is the main performance measure, the architectures seem to be robust to changes in decoder depth. Also, a direct comparison of loss and accuracy shows that the transformer outperforms the LSTM in how quickly it reduces the test loss and gains in test accuracy.



(a) Test loss of the models. (b) Test accuracy of the models.

Figure 2: Comparison of results on test data for the two competitive models with encoder layers fixed to one. 'D' stands for the number of decoder layers. Batch-size = 64

In the second robustness test, decoder layers are fixed to three, and encoder layers varied, with results plotted in Figure 3. The difference in performance is significant for both architectures for the loss but not accuracy, except for the three-encoder layer LSTM. This indicates that in formula encoding, some relevant information is retrieved by the additional encoder layers. However, the difference in performance is very small. Effect-wise, the most important factor seems to be the choice of architecture where the transformer outperforms the LSTM under all conditions.



(a) Test loss of the models. (b) Test accuracy history.

Figure 3: LSTM and transformer results for different numbers of encoder layers, with decoder layers fixed to one. 'E' stands for the number of encoder layers. Batch-size = 64

5.2. Produced Formulas

For each of the three architectures with one encoder and three decoder layers, 50 random derivations were sampled. In the case of the transformer, there were four attention heads in the encoder and decoder. Derivations were produced for the test data, and I checked them for correctness by hand.

The feedforward network baseline could only learn a simple bracket language. Here, all 50 outputs were no derivations; neither the premises nor the conclusions were correct. The bracket language had some similarities with $\mathcal{L}$-formulas that also contained brackets, opened and closed in even numbers, with logical connectives and propositional variables within. These were not yet well-formed formulas but showed the feedforward network's ability to learn structure, even though it did not have recurrence. One example of these bracket expressions is:

$$q, (s \wedge (p \rightarrow s)) \vdash (q \wedge (q \rightarrow (s \wedge (p \rightarrow s)))) \quad \text{(Input)}$$
$$((\rightarrow \wedge)(s\wedge))((ss((s \wedge (s \wedge s)))) \quad \text{(Output)}$$

The LSTM above the simple bracket regularities put out well-formed formulas about half the time. Outputs were, in 46% of cases, sequences of only well-formed formulas of $\mathcal{L}$. In the case of the other 54% of non-well-formed formulas, 75.6% again contained expressions of a bracket language like the ones the feedforward network produced. However, in all cases of well-formed formulas, propositional variables are exchanged as in:

$$(t \rightarrow r), s \vdash ((t \rightarrow r) \wedge (r \vee (t \rightarrow r))) \quad \text{(Input)}$$
$$(t \rightarrow s), p, (p \vee (p \rightarrow p)), ((t \rightarrow s) \wedge (t \vee (t \rightarrow s))) \quad \text{(Output)}$$

There was no observable regularity to the replacement as if the trained model could not preserve the correct propositional variables. Still, in 40% of output derivations, the premises matched the premises in the input up to the replacement of propositional variables by random (the same propositional variables were replaced by different ones in the output, indicating no systematicity). Here, the output derivations were correct up to the random replacement of variables.

The transformer managed to reproduce the input premises in the output derivations in all 50 cases. Conclusions matched in 86% of cases. There was no replacement of variables present as in the LSTM. 70% of derivations were correct, for example:

$$s, ((\neg t) \wedge t) \vdash (r \vee (s \vee s)) \quad \text{(Input)}$$
$$s, ((\neg t) \wedge t), (s \vee s), (r \vee (s \vee s)) \quad \text{(Output)}$$

Errors in derivations happened in several forms. The most common was the model falling back to a simple bracket language in 46.6% of incorrect derivations. 26.6% of incorrect derivations were due to some propositional variable being deleted at some step in the derivation. In 20% of incorrect derivations, the wrong derivation rule was used. No regularity was observable with respect to derivation rules needing to be applied to derive a conclusion and errors happening at this sample size. This could, however, be the basis for further inquiry, where the rules that needed to be applied are written to the test dataset. Then, the correlation between these rules and low accuracy for a given output could be measured.

6. Discussion

The feedforward network learned a bracket language, where brackets are closed in an even manner. For this, a context-free grammar is also necessary because a regular grammar would not be capable of keeping track of the brackets already opened. Given that all the architectures considered were trained to become models that could produce context-free languages, it can be said that they are at least part of the set of push-down automata. This seems to go against the findings of [6], which were that transformers have problems with even regular tasks. However, the differences between their definition of rule-following and mine must be considered here. Because, for them, rule-following amounts to

deriving longer strings with given rules, they tested the models on deriving longer strings than they had to derive during training. In my case, training and test outputs were sampled from the same dataset. They were only different with respect to the input, the premise-conclusion pairs. Thus, there is no contradiction between their results and mine.

The main reason [6], named for the bad performance of the transformer architecture, was its permutation invariance, which was only augmented by positional encoding. However, given the surveyed literature and my results, this seems only to be a problem of deriving longer strings. But also there, the generality of the results by Delétang et al. can be taken into question as the meta-learning challenge incorporated also the derivation of longer strings. Even still, positional encoding preserves syntactic structural information better than recurrence in LSTMs. This is the case even though the transformer used the same feedforward networks as the baseline.

References

[1] D. D. McCracken, E. D. Reilly, Backus-Naur form (BNF), John Wiley and Sons Ltd., GBR, 2003, p. 129–131.

[2] Y. Chen, S. Gilroy, A. Maletti, J. May, K. Knight, Recurrent neural networks as weighted language recognizers, in: Proceedings of the 2018 Conference of the North American Chapter, Human Language Technologies, 2018.

[3] J. Pérez, J. Marinkovic, P. Barceló, On the turing completeness of modern neural network architectures, 2019.

[4] J. Pérez, P. Barceló, J. Marinkovic, Attention is turing-complete, J. Mach. Learn. Res (2021).

[5] H. Siegelmann, E. Sontag, Analog computation via neural nets, Theoretical Computer Science 131 (1994) 331–360.

[6] G. Delétang, A. Ruoss, J. Grau-Moya, T. Genewein, K. Li, E. Wenliang, C. Catt, M. Cundy, S. Hutter, J. Legg, P. Veness, Ortega, Neural networks and the chomsky hierarchy, 2023.