Length Generalization on Multi-Digit Integer Addition with Transformers

Imran Ibrahimli
imran.ibrahimli@studium.uni-hamburg.de
Knowledge Technology, WTM
Dept. Informatik
Universität Hamburg

Advisors:
Prof. Dr. Stefan Wermter, Dr. Jae Hee Lee

19.11.2024

Outline

Outline

Motivation

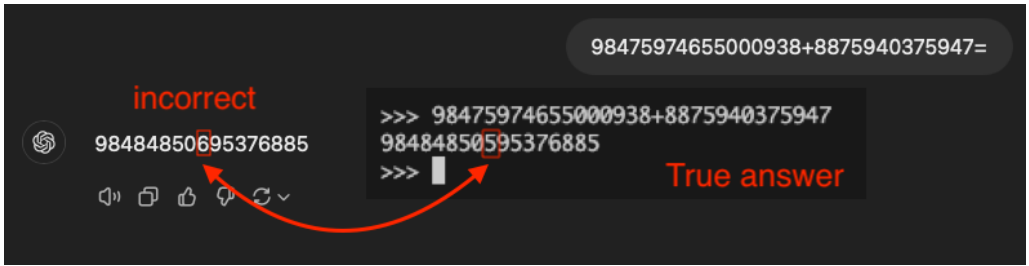
- Transformer models are being adopted across many domains from language modeling to robotics.
- For effective applications, models need to learn generalizing operations or algorithms from data.
- Generalization enhances robustness and trust in AI systems.

Motivation (cont.)

- Examine a simple toy task: integer addition.
- We don't need transformers to perform addition.
- But we care if they can learn "simple" algorithms from examples.
- Length-generalizable multi-digit addition is an open problem for transformers.
- Actively worked on with many papers in NeurIPS, ICLR, etc.

Motivation (cont.)

- Example: even SOTA model GPT-4o struggles with addition.



Motivation (cont.)

- Why is this important?
- Maybe just guardrail models by hard-coding some rules?
- Not generally, because we would not train and use the models, if we could hard-code the rules in the first place.
- We want to understand the limitations of the models and improve them.

Problem Statement

- Focus on standard decoder-only transformers with absolute positional encodings.
- Try to improve generalization without altering model architecture or task-specific modifications.
- Explore data formatting and training data diversity.

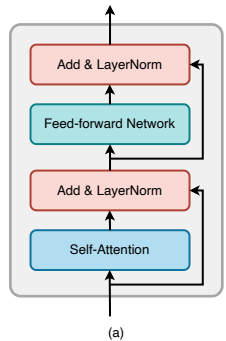
Research Questions

1. Why do transformers with absolute positional encodings fail to generalize integer addition to longer sequences?
2. How does the inclusion of sub-task data influence the model's compositionality and length generalization capabilities?

Outline

Transformers

- Utilize self-attention to process sequential data.
- Focus on decoder-only models (b).
- Capable of capturing long-range dependencies without recurrence.



Positional Encodings (PE)

- No explicit positional awareness in transformers.
- PEs inject sequence order information.
- Can be absolute or relative.
  - Absolute PEs:
    `0, 1, 2, 3, ...`
  - Relative PEs:
    `..., -3, -2, -1, 0, 1, 2, 3, ...`

Absolute Positional Encoding

- Adds fixed positional information to input embeddings.
- Sinusoidal absolute PEs introduced by Vaswani et al. (2017):

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Outline

Challenges in Length Generalization

- Positional encoding plays a significant role.
- Kazemnejad et al. (2023) find even models without positional encodings can length generalize somewhat.
- Experiments in this thesis do not support these findings.

Issues with Positional Encodings

- Models struggle to select relevant tokens based on position in longer sequences.
- Failure to **align digits** without explicit positional cues.
- Confirmed by multiple studies (Shen et al. 2023; Zhao et al. 2024).

Improving Length Generalization

- Reversing answer and training with scratchpad (Lee et al. 2024).
- Randomized PEs (Ruoss et al. 2023).
- Inclusion of random spaces in input (Shen et al. 2023).
- Index hints `a1b2c3+a4b5c6=a5b7c9` (Y. Zhou et al. 2024)
- Task-specific PEs, e.g. Abacus (McLeish et al. 2024).

Length Generalization Ratio

- Ratio of the length of solved test problems to training lengths.
- 1.1x ratio achieved by Shen et al. (2023).
- 1.125x by Kazemnejad et al. (2023).
- 1.5x by H. Zhou et al. (2023).
- 2.5x with FIRE and index hints (Y. Zhou et al. 2024).
- **6x** achieved with Abacus encoding (McLeish et al. 2024).

Abacus Encoding

- Encode digit position relative to the start of the number.
- Need to reverse integers
- Use sinusoidal PE with indices specified by the Abacus encoding.
- Combined with architectural modifications like input injection and recurrent layers.
- Trained on up to 20 digits, generalizes to 120 digits.

Figure: Source: McLeish et al. (2024)

Conclusions from Related Work

- No common benchmark $\rightarrow$ hard to compare methods.
- Positional encodings and data formatting significantly impact length generalization.
- Task-specific design can boost generalization to unseen lengths.
- Some diverge from standard decoder unsupervised training.

Limitations of Existing Methods

- Some successful methods lose sight of the original task:
  It's not about addition, but about model capabilities.
- Task-specific modifications are not always feasible.
- For the same reason e.g. adding index hints is "hacky."
- Desire for simpler methods without altering model architecture.

Outline

Overview of Approach

- Investigate various positional encodings and data formatting.
- Examine the impact of sub-task data on compositionality.
- Apply mechanistic interpretability to analyze models.

Data Formatting Techniques

Technique	Example
Standard format	`123+456=`
Reversed operands	`321+654=`
Random spaces	`1 23 +4 5 6=`
Zero-padding	`00123+00456=`
Scratchpad	`$567+789=765+987;` `c=0,7+0+0=7,c=0;` `6+9+0=5,c=1;` `5+8+1=4,c=1;` `0+7+1=8,c=0\|8457$`

Reproducing Baseline Failure: Results

- High accuracy on training lengths (1 and 3 digits).
- Significant drop in accuracy on unseen lengths (2 and 4 digits).

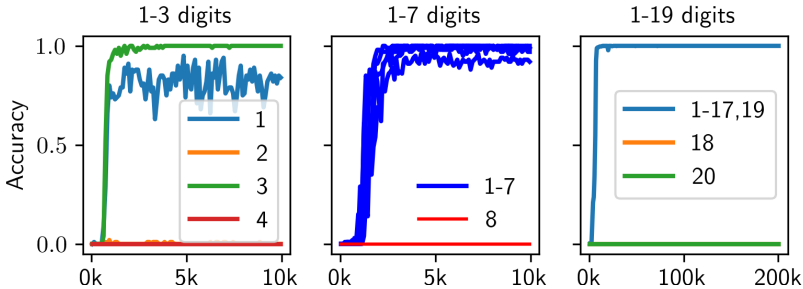


Figure: Baseline performance over training epochs.

Reproducing Baseline Failure: Analysis

- Model relies on absolute positions seen during training.
- Fails to align digits correctly for unseen lengths.
- Highlights limitations of absolute positional encoding.

Extending Baseline Observations

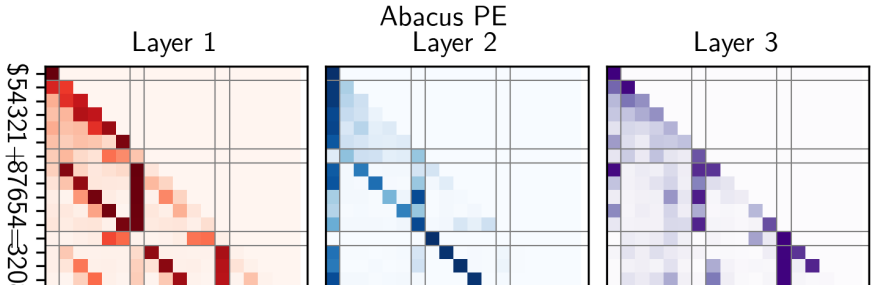
- Trained on 1-7 digit additions.
- Tested on 8-digit additions.
- Consistent failure to generalize to longer sequences.

Assessing Generalization Across Lengths

- Trained on 1-19 digit additions (excluding 18-digit cases).
- Tested on 1-20 digit additions.
- No significant generalization to unseen lengths.

Impact of Data Formatting and Dataset Size

- Introduced random spaces in input sequences.
- Trained on 1-7 and 9-digit additions.
- Tested on 8 and 10-13 digits.
- Slight improvement in length generalization observed.



Introducing Sub-Task Data

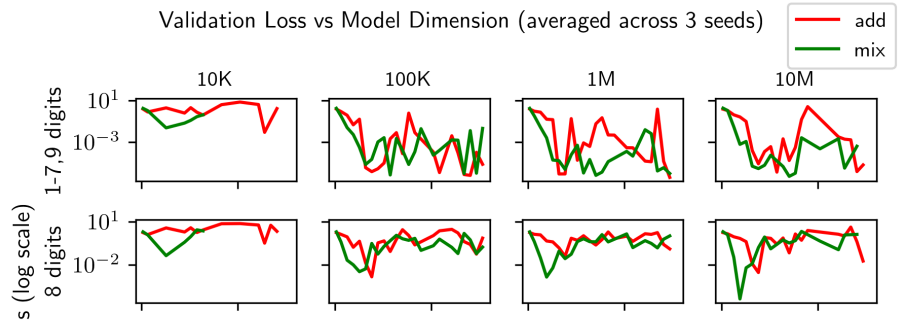
- Sub-tasks: carry detection, modular addition, reversing, digit alignment.
- Used task-specific prefixes to distinguish tasks.
- Trained models on combined data.

Table: Examples of Sub-Task Data

Sub-Task	Example
Modular Addition	`mod_add: 7 + 8 = 5`
Carry Detection	`carry: 7 + 8 = 1`
Reversing	`reverse: 123 = 321`
Digit Alignment	`align: 123`

Effect on Compositionality

- Analyzed the model's ability to compose functions.
- Marginal improvements in length generalization observed.
- Smaller models benefited more from sub-task training.

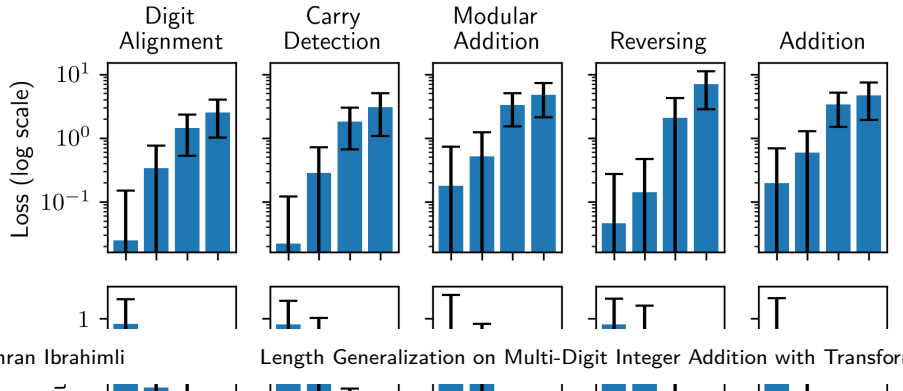


Comprehensive Analysis Across Scales

- Explored model dimensions from 64 to 1536.
- Dataset sizes from 10K to 10M examples.
- Compared models trained with and without sub-task data.

Sub-Task Difficulty Analysis

- Evaluated test loss and accuracy for sub-tasks.
- Modular addition and digit alignment were most challenging.
- Significant variability in performance across sub-tasks.



Mechanistic Interpretability

- Analyzed attention maps of models.
- Models with Abacus encoding showed crisp attention patterns.
- Models with absolute positional encoding had diffuse attention.
- Indicates challenges in digit alignment and carry propagation.

Outline

Summary of Findings

- Transformer models with absolute positional encodings fail to generalize due to rigid digit alignment.
- Abacus positional encoding significantly improves length generalization.
- Data formatting techniques offer marginal improvements.
- Sub-task data enhances compositionality, especially in smaller models.
- Mechanistic interpretability reveals differences in internal representations.

Future Work

- Develop positional encodings that generalize without task-specific modifications.
- Explore other data augmentation techniques to improve generalization.
- Apply mechanistic interpretability to more complex tasks.
- Investigate the impact of model architecture variations on length generalization.

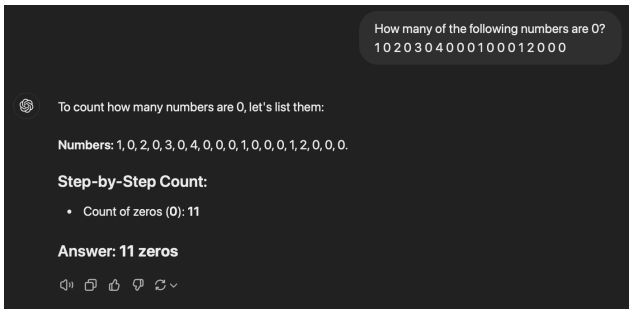
Questions

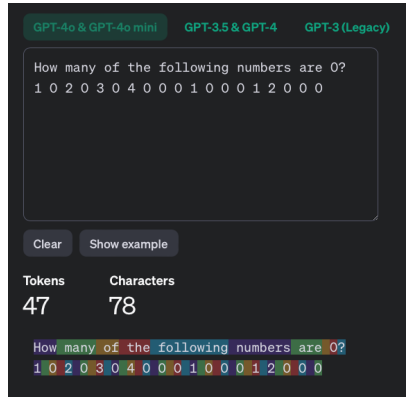
Thank you for your attention!

Questions?

Appendix: Counting Failure

- Example: SOTA model GPT-4o struggles with simple counting.





Self-Attention Mechanism

- Computes attention weights between all token pairs.
- Allows the model to focus on relevant parts of the sequence.
- Attention function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- Where Q, K, V are query, key, and value matrices.

Training and Inference