Report: Learning addition with Small Transformer

Github link:
https://github.com/sultan-hassan/AISafety/tree/main/learning_addition

1 - Task setup:

The goal is to train a small transformer to perform base-10 addition of 2 k-digit integers with k=3.

- **Model**: A decoder-only GPT transformer with 6 layers, 8 heads, 256 embedding dimensions.
- **Tokenizer**: Vocab: 0-9, +, =
- **Format Strategy**: The Reverse-Output Trick. Instead of 123+456=579, the model was trained on 123+456=975.
  - Justification: Transformers generate outputs from left-to-right. Standard addition requires calculating carries from right-to-left. Reversing the output aligns the model's generation order.

2 - Data design:

We have tried different strategies:

1 - uniform sampling (same k): e.g. 408+458=668

2 - Uniform Mixed-Length (multiple k): e.g. 35+8=34 and 1+399=004

3 - Zero-Padding (Fixed Geometry): all input has 4 digits, e.g. 0011+0022 = 33000

3 - Training details:

- Trained locally and runtime is very fast (< min).
- Default optimizer is AdamW (lr=5e−4, weight decay 0.01)
- Batch size 64, training epochs 20
- Default cross entropy loss

4 - Evaluation suites:

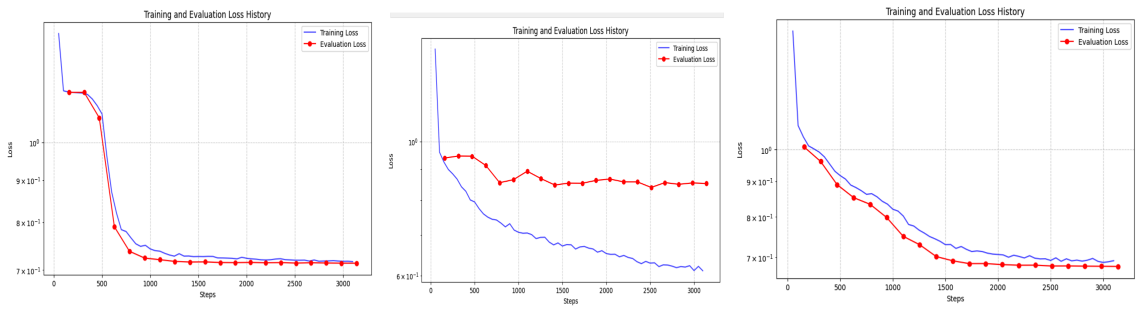
The metrics below used to quantify both in-distribution (with k=3) and out-of-distribution samples (with different k=4)

- Loss evolution to test whether the model is under/over-fitting
- Exact match between target and prediction
- Digit-wise accuracy: This measures the proximity of the prediction to the target. If the model predicts the first several digits accurately, it indicates the model is beginning to converge on the correct logic.
- Mean Absolute Error (MAE): This measures *magnitude* of failure

- Average Edit Distance: This treats numbers as string counting how many keystrokes needed to fix the answer.
- No curry accuracy: This quantifies basic retrieval and simple mapping (e.g. 12+23 =35).
- With curry accuracy: This tests reasoning and working memory (e.g. 9+5 = 14)

4 - Results:

- Training with padded examples results in the lowest loss as shown in the figure below (left to right: data strategy 1-3, see train.ipynb notebook in the github link above).



- Table: Summary of other metrics in the following format: ID (OOD) metric

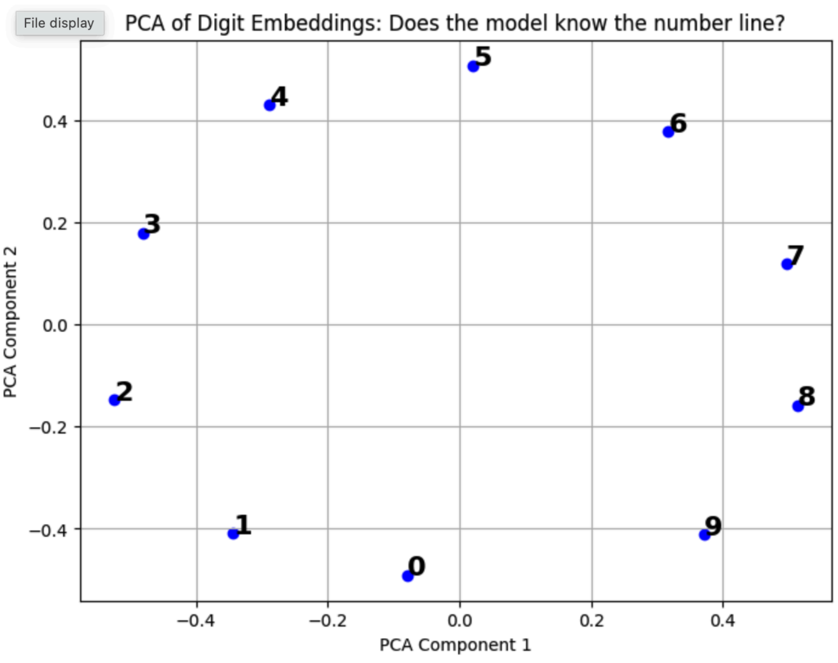
Data strategy	Exact match	Digit-wise	MAE	Edit distance	No curry	with curry
Uniform (same k)	100% (0.0%)	100% (3.35%)	0.0 (10880.65)	0.0 (3.73)	100% (0.0%)	100% (0.0%)
Uniform (diff k)	100% (0.0%)	100% (3.35%)	0.0 (10940.77)	0.0 (4.62)	100% (0.0%)	100% (0.0%)
padded	99.50% (0.0%)	99.88% (**48.64%**)	0.5 (**10287.83**)	0.01 (**2.55**)	100% (0.0%)	99.15% (0.0%)

This table shows that training with zero padded datasets has better accuracy for predicting OODs.

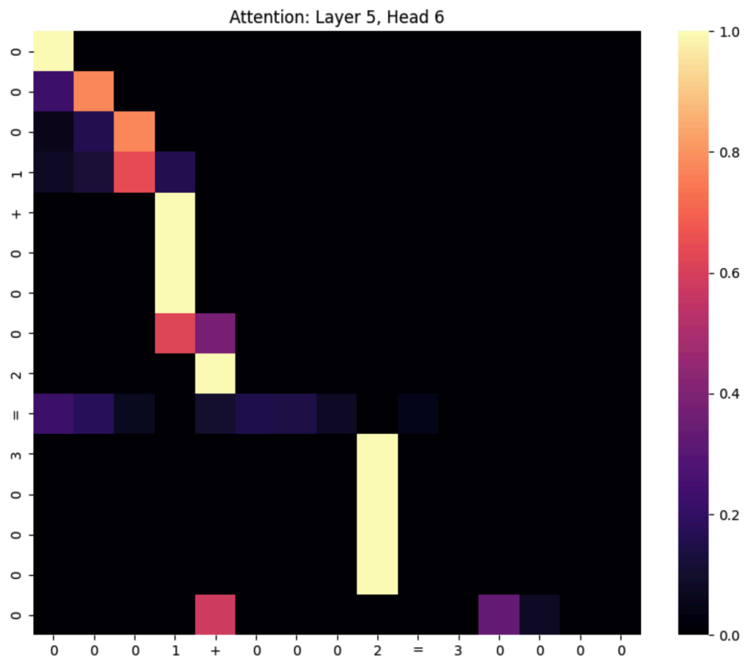
4 - Interpretation:

Using mechanistic interpretability approaches, we have attempted to understand what the model has learned.

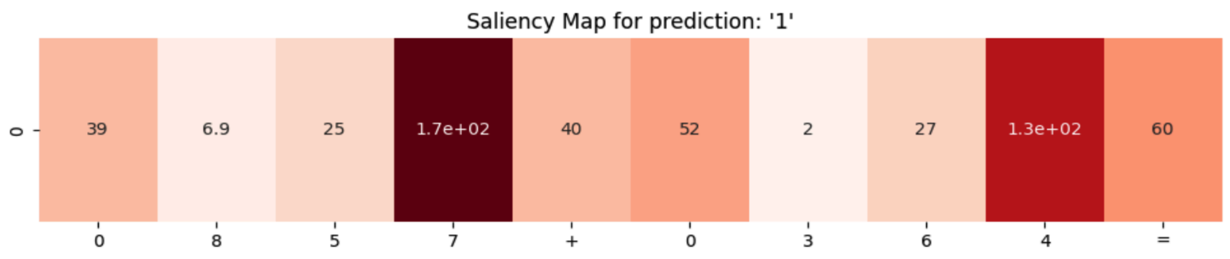
1 - Embedding geometry (figure below): PCA projection of the digit embedding (0-9) revealed a perfect number circle. This shows the model has learned the periodic structure of the numbers (0→ 1→ 2→ ..). This is similar to the finding by Nanda's Grokking paper where they have shown the same by decomposing the model activations into fourier space.



2 - Visualization attention heads for some simple addition (0001+0002+30000, figure below): This shows the most intense bright spot on x-axis is digit 2, which starts from 3 on Y-axis. This shows the model uses positional look-up to perform addition (similar maps are seen for other input numbers).



3 - Saliency mapping with input “0857+0364=” to prediction ‘1’ (figure below). This shows that 7 and 4 have the highest values, a proof of column wise addition.



4 - Ablation:

- Head ablation: removing a single head has zero impact on the accuracy, indicating redundancy.
- Layer ablation: removing any single layer dropped the accuracy to zero, indicating strong serial dependency throughout the network.

In summary: The model

- Does not generalise to out of distribution samples, indicating poor reasoning.
- Learned periodic structures, patterns, and positional algorithms, rather than a universal calculator.

Next steps:

- Testing the viability of using chain-of-thought datasets to robustly monitor the reasoning accuracy
- Testing different architectures, and larger models.