Towards Interpretability of 5 digit addition[1]

Philip Quirke

Abstract

A 1L Transformer was trained on the problem of 5 digit addition by Neel Nanda et al. When trained on 700 data points the model demonstrated clear grokking. When the loss is decomposed into the loss on each of the 6 digits in the sum, there are separate phase changes in each digit. Further, the ordering of the phase changes is not stable between runs.

This paper details a hypothesis for the internal structure of the model that may explain the observed variability & proposes specific testing to confirm (or not) the hypothesis.

Keywords:Mechanistic interpretability, AI safety, 5 digit addition

1. Introduction

The [2301.05217] Progress measures for grokking via mechanistic interpretability paper by Neel Nanda et al, argued that progress measures can be found via mechanistic interpretability: reverse-engineering learned behaviors into their individual components. Section 5.2 references a 1-layer 5-digit addition model. Figure 29 displayed the "Per Token Train/Test Loss" but left unexplained the differences in curves between the digits.

A related post A Mechanistic Interpretability Analysis of Grokking speculated that "Phases Changes are Everywhere" and for the 5 digit addition model noted that:

> We can decompose the loss into the sum of components - the loss on each of the 6 digits in the sum. When we do this, we observe separate phase changes in each digit, resulting in the many small non-convexities in the overall loss curve.

- The ordering of phase changes is not stable between runs, though token 0 and token 1 tend to be first[14]
- This isn't specific to 5 digit, eg 15 digit addition shows 16 separate phase changes

[1] Research conducted at the Apart Research Alignment Jam #10 (Interpretability 3.0), 2023 (see https://alignmentjam.com/jam/interpretability)

Per digit loss for each token in the output for 5 digit addition, trained on infinite data, plotted against the overall loss

This paper details a hypothetical model for the internals of the 5 Digit Addition neural net. This model's alignment to the observed phenomenon is discussed. Specific future experiments are recommended that would provide evidence for or against this hypothesis.

2. Methods

2.1 Motivation: Speed JigSaws

A friend once proposed a way to minimize the time to complete a jigsaw. A variant of this proposal has applicability to the 5 Digit Addition model.

The "speed jigsaw" proposal required 60 people to stand around a table each holding one (corner, edge or internal) piece of the 60 piece jigsaw. Each piece has a clear orientation, so rotating pieces is not required. Each person can only put their piece down if they are certain they know where it goes. No-one knows ahead of time what the jigsaw picture is. If a piece adjacent to the piece a person is holding has already been placed on the table, the person can correctly place their piece down.

When the timekeeper says 'Go!' the 4 people holding corner pieces place them and step away from the table. Then 8 people holding corner-adjacent edge pieces can place them and step away. Then 8 people can place additional edge pieces, and at the same time 4 people can place internal pieces (diagonally from the corner pieces). The process continues, and the last piece is placed near the middle of the jigsaw.

The similarities with the 5 Digit Addition model are:

- The 60 independent people represent 60 independent neurons
- There is no pre-agreed overall strategy or communication or co-ordination between the 60 people - just a few "rules of the game" to obey.
- Each person is assigned one specific and very simple task - place their piece. Some tasks (place a corner piece) are easier than others (place an internal piece)
- There is inherent time ordering in most tasks: Most tasks can only occur after their predecessor task(s).
- The people work in parallel - at each time slice (Epoch) each person with a piece in their hand does some computation and may place a piece.
- The jigsaw progresses over time towards completion.
- The overall process may be optimal and is certainly very time efficient.

2.2 Assumptions and Approach to Analysis

This analysis of the 5 Digit Addition model assumes:

- The algorithm uses a small number of as-simple-as-possible "base" tasks that have no predecessor
- The algorithm uses a small number of "compound" tasks that are composed of and reliant on other base predecessor task(s)
- Compound tasks can only be successfully trained, after their base predecessors have been (mostly) trained, and so there is an inherent training time ordering
- For the 5 Digit Addition model to calculate say the 3rd column addition, we need the "carry 1" result from the 2nd column addition, and so there is an inherent training ordering between the columns.
- The algorithm is very efficient and very simple & the tasks can be intuited
- Base tasks are simpler than compound tasks and so are discovered earlier in training runs.
- Aligned to Grokking, memorisation tasks (not detailed in this paper) may dominate early in training, before the "better" base and compound tasks are discovered later in training, and over time come to dominate the model.

These assumptions lead to a model which has expected implications which can be tested via experimentation. This will be detailed later.

2.3 Base Tasks

There are 3 "base" tasks in the 5 digit (column) addition algorithm. They all relate to a single column addition. They are independent.

They are:

- **BaseAdd:** Add two digits in a column, ignoring carry over from previous column and carry forward to next column. So 1+2=3, 5+6=1, 7+9=6.
- **ForceCarry:** Does adding two digits in this column force a "carry 1" to the next column? Ignores carry over from previous column. So 4+5=False, 5+5=True, 6+8=True.

- **Sum9:** Do the two digits in this column add to 9? So 0+9=True, 4+5=True, 3+4=False

BaseAdd
This simple base task should be discovered early and give good results early in training:
- In column 2 to 5, this task will give the correct answer for the column 50% of the time (when the training example does not carry over 1 from the previous column).
- In column 1, this task will give the correct answer 100% of the time

There are 100 unique possible cases, in each of the 5 columns.

ForceCarry
This is an enabler task that evaluates state. It must be discovered in training. It does not improve training results immediately upon discovered. A separate compound task (described later) is needed to leverage ForceCarry.

There are 100 unique possible cases, in each of the 5 columns.

Sum9
This is an enabler task that evaluates state. It must be discovered in training. It does not improve training results immediately upon discovery. A separate compound task (described later) is needed to leverage Sum9.

There are 10 (ten) unique possible cases, in each of the 5 columns.

2.4 Compound Tasks
There are two compound tasks.

They are:
- **UseForceCarry:** If ForceCarry is true for the previous column, add 1 to this column (modulus 10). Ignore possible knock on effects on the next column.
- **UseSum9:** If ForceCarry is true for the previous column, and Sum9 is true for this column, add 1 to the next column. This may have a knock on effect of changing the value of the ForceCarry on the next column.

These compound tasks:
- Need a base task to exist and provide good data, before they are useful. So they add value later in the training run. (This is an inherent time ordering)
- Propagate an effect forward from one column to the next higher column (This is an inherent column ordering).
- Are independent of each other.

UseForceCarry
This compound task relies on the base task ForceCarry.
If ForceCarry is accurate, UseForceCarry improves digit-level training results:
- In column 2 to 5, this task will improve the answer accuracy by 45%
- In column 1, this task is not applicable.

UseForceCarry will occur around 10 times as frequently as UseSum9.

UseSum9
This compound task relies on the base task Sum9.
If Sum9 is accurate, UseSum9 improves digit-level training results:

- In column 2 to 5, this task will improve the answer accuracy by 5%
- In column 1, this task is not applicable.

2.5 Additional Assumption

While there may be alternative formulations of the base and compound tasks to do addition, I claim (without proof) these would be fundamentally equivalent to the above set of tasks, because the above tasks appear to be the minimal and necessary set of "skills" necessary to learn to do addition.

The tasks that increase answer accuracy (BaseAdd, UseForceCarry and UseSum9) are independent. They could be discovered in any order. But BaseAdd, as the only "base" task, will likely be discovered before the compound tasks.

2.6 Example Task & column algorithm discovery orderings

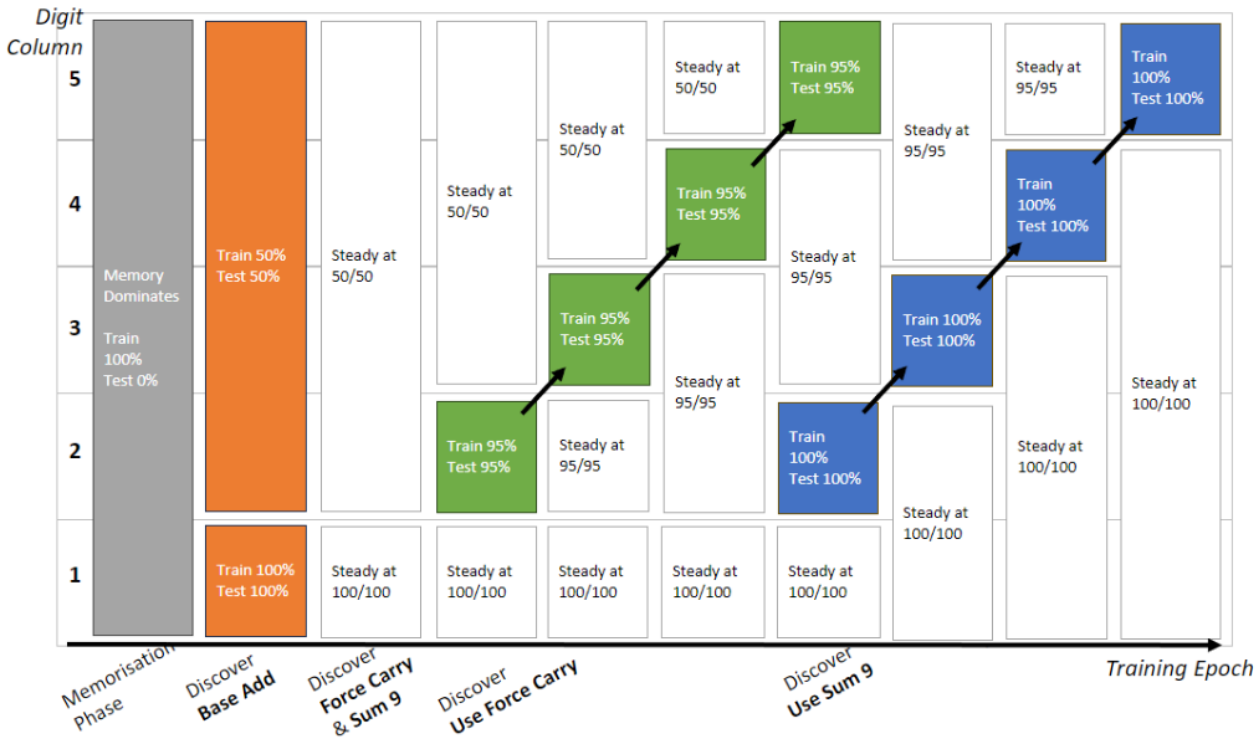
The following diagram outlines a simplified view of how the algorithm works:

- The vertical axis represents each digit column.
- The horizontal axis is training epoch (time).
- The diagram body includes estimates of the cumulative impact on Training & Testing percentage correct per digit the rules deliver.

The body also shows the inherent "delay" in each digit column providing good data to the next digit column.

1. The task discovery order shown in the diagram is not the only possible one but is an easy one to understand:
2. The model initially does memorisation
3. BaseAdd is discovered and applied giving 50% accuracy
4. ForceCarry and Sum9 "state information" are discovered but give no more accuracy (yet)
5. UseForceCarry is discovered and applied to successive digit columns, adding 45% accuracy
6. Finally UseSum9 is discovered and applied to successive digit columns, adding 5% accuracy

In reality all the tasks may be discovered and be refined at the same time (as they are independent), rather than sequentially as shown in the diagram (for simplicity).



2.7 Alignment to Problem Statement Observations

This algorithm explains the observations in the problem statement:

Statement: The ordering of phase changes is not stable...

This algorithm includes 3 independent tasks that deliver accuracy improvements (BaseAdd, UseForceCarry, UseSum9) visible in loss curves. These tasks can be discovered by the neural net in any order, leading to changes in the ordering of phase changes.

(ToDo: This needs more investigation. Can the rule discovery order differ per digit? What would be the impact of a different order on training efficiency?)

Statement: ... though Token 0 and Token 1 tend to be first.

The first digit (Token 0) sum can be calculated using just BaseAdd. This simplicity would naturally lead to the first digit usually being grokked first. The second digit (Token 1) sum can be calculated using just BaseAdd, and the ForceCarry and UseCarry tasks as they relate to the first digit only. The third and successive digits have a longer dependency change & so will naturally tend to take longer to grokk.

Statement: ... 15 digit addition shows 16 phase changes.

The BaseAdd task delivers an accuracy improvement of 50%. This may be applied to all digits roughly simultaneously giving 1 visible phase change.

The UseForceCarry task delivers an accuracy improvement of 45%, but the dependency chain means that for 15 digits there are 15 phase changes, separated in time.

This gives the 16 phase changes

(The UseSum9 task delivers an accuracy improvement of 5%. This improvement is likely too small to be visible as a phase change.

2.8 Collecting Evidence of Existence of Tasks

The author's low level of knowledge of neural net analysis and the Transformer Lens framework, together with time pressures, constrained from performing these experiments.

These experiments (none of which have been carried out to date) would provide evidence in favor or against the hypothesis:

ForceCarry and UseForceCarry task identification

Create test data where all digits force a carry e.g. 33333+88888, 44444+77777, 34567+87654. The neurons that activate most strongly are likely ForceCarry and UseForceCarry neurons.

Create test data with a carry occurs only in say the 3rd digit e.g. 22222+44944. The neurons that activate most strongly may be ForceCarry and UseForceCarry neurons for digit 3.

Set the activation on these neurons to zero, rerun the same test data, and check that the carry does not occur.

Sum9 task identification

Create test data where there is no overflow but all the digits add to 9 e.g. 11111+88888, 22222+77777, 12345+87654. The neurons that activate most strongly are likely Sum9 neurons.

Create test data with no overflow, and where only say the 3rd digit sums to 9 e.g. 11111+22822, 22222+44744. The neurons that activate most strongly may be Sum9 neurons for digit 3.

UseSum9 task identification

Create test data where there is an overflow in say digit 2, and the digits in column 3 add to 9, so that an overflow is forced into column 4 e.g. 00550+00450=01000, 22662+22342=45004. The neurons that activate most strongly are likely Sum9 and UseSum9 neurons for digit 2.

Set the activation on these neurons to zero, rerun the same test data, and check that the carry does not occur.

3. Results

The "results" of the analysis are:

- The documented hypothetical model of the 1-layer 5 Digit Addition neural net.

- The documented (pending) experiments

4. Discussion and Conclusion

Reverse-engineering learned behaviors of neural nets into their individual components is non-trivial for several reason including:

- Human and neural net "coding" approaches to solving the same problem differ greatly
- Neural nets cannot explain themselves
- Neural nets "try out" many alternative approaches, concurrently, during training
- Neural nets training is a stochastic experimentation process

However, this paper demonstrates a potentially successful, potentially generalizable approach, based on reframing the human approach to the problem (in this case how humans add two numbers) in terms of:

- As-small-as-possible "base" cognition tasks that are learnt concurrently, independently and stochastically by the neural net
- Composing base tasks together to yield slightly-higher-level "compound" cognition tasks; again learnt concurrently, independently and stochastically,
- Considering time-lag dependencies between tasks inherent in the problem space
- Considering relative frequency & benefit of each task type
- Generating a coherent, testable model of the neural net internals.

This approach may provide an additional tool in the mechanistic interpretability toolbox. More testing is needed.

5. References

[2301.05217] Progress measures for grokking via mechanistic interpretability, Neel Nanda et al.