
NE
E
SE
S
SW
W
NW
N

[Redwood Research] Causal Scrubbing

Causal Scrubbing: a method for rigorously testing interpretability hypotheses [Redwood Research]

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

* *Authors sorted alphabetically.*

Summary: This post introduces causal scrubbing, a principled approach for evaluating the quality of mechanistic interpretations. The key idea behind causal scrubbing is to test interpretability hypotheses via *behavior-preserving resampling ablations*. We apply this method to develop a refined understanding of how a small language model implements induction and how an algorithmic model correctly classifies if a sequence of parentheses is balanced.

1 Introduction

A question that all mechanistic interpretability work must answer is, "how well does this interpretation explain the phenomenon being studied?". In the many recent papers in mechanistic interpretability, researchers have generally relied on ad-hoc methods to evaluate the quality of interpretations.[1]

This *ad hoc* nature of existing evaluation methods poses a serious challenge for scaling up mechanistic interpretability. Currently, to evaluate the quality of a particular research result, we need to deeply understand both the interpretation and the phenomenon being explained, and then apply researcher judgment. Ideally, we'd like to find the interpretability equivalent of property-based testing—automatically checking the correctness of interpretations, instead of relying on grit and researcher judgment. More systematic procedures would also help us scale-up interpretability efforts to larger models, behaviors with subtler effects, and to larger teams of researchers. To help with these efforts, we want a procedure that is both powerful enough to finely distinguish better interpretations from worse ones, and general enough to be applied to complex interpretations.

In this work, we propose **causal scrubbing**, a systematic ablation method for testing precisely stated hypotheses about how a particular neural network[2] implements a behavior on a dataset. Specifically, given an informal hypothesis about which parts of a model implement the intermediate calculations required for a behavior, we convert this to a formal correspondence between a computational graph for the model and a human-interpretable computational graph. Then, causal scrubbing starts from the output and recursively finds all of the invariances of parts of the neural network that are implied by the hypothesis, and then replaces the activations of the neural network with the *maximum entropy*[3] distribution subject to certain natural constraints implied by the hypothesis and the data distribution. We then measure how well the scrubbed model implements the specific behavior.[4] Insofar as the hypothesis explains the behavior on the dataset, the model's performance should be unchanged.

Unlike previous approaches that were specific to particular applications, causal scrubbing aims to work on a large class of interpretability hypotheses, including almost all hypotheses interpretability researchers propose in practice (that we're aware of). Because the tests proposed by causal scrubbing are mechanically derived from the proposed hypothesis, causal scrubbing can be incorporated "in the inner loop" of interpretability research. For

example, starting from a hypothesis that makes very broad claims about how the model works and thus is consistent with the model's behavior on the data, we can iteratively make hypotheses that make more specific claims while monitoring how well the new hypotheses explain model behavior. We demonstrate two applications of this approach in later posts: first on a parenthesis balancer checker, then on the induction heads in a two-layer attention-only language model.

We see our contributions as the following:

1. We formalize a notion of interpretability hypotheses that can represent a large, natural class of mechanistic interpretations;
2. We propose an algorithm, *causal scrubbing*, that tests hypotheses by systematically replacing activations in all ways that the hypothesis implies should not affect performance.
3. We demonstrate the practical value of this approach by using it to investigate two interpretability hypotheses for small transformers trained in different domains.

This is the main post in a four post sequence, and covers the most important content:

- What is causal scrubbing? Why do we think it's more principled than other methods? (sections 2-4)
- A summary of our results from applying causal scrubbing (section 5)
- Discussion: Applications, Limitations, Future work (sections 6 and 7).

In addition, there are three posts with information of less general interest. The first is a series of appendices to the content of this post. Then, a pair of posts covers the details of what we discovered applying causal scrubbing to a paren-balance checker and induction in a small language model.[5] They are collected in a sequence here.

1.1 Related work

Ablations for Model Interpretability: One commonly used technique in mechanistic interpretability is the "ablate, then measure" approach. Specifically, for interpretations that aim to explain why the model achieves low loss, it's standard to remove parts that the interpretation identifies as important and check that model performance suffers, or to remove unimportant parts and check that model performance is unaffected. For example, in Nanda and Lieberum's Grokking work, to verify the claim that the model uses certain key frequencies to compute the correct answer to modular addition questions, the authors confirm that zero ablating the key frequencies greatly increases loss, while zero ablating random other frequencies has no effect on loss. In Anthropic's Induction Head paper, they remove the induction heads and observe that this reduces the ability of models to perform in-context learning. In the IOI mechanistic interpretability project, the authors define the behavior of a transformer subcircuit by mean-ablating everything except the nodes from the circuit. This is used to formulate criteria for validating that the proposed circuit preserves the behavior they investigate and includes all the redundant nodes performing a similar role.

Causal scrubbing can be thought of as a generalized form of the "ablate, then measure" methodology.[6] However, unlike the standard zero and mean ablations, we ablate modules by resampling activations from *other* inputs (which we'll justify in the next post). In this work, we also apply causal scrubbing to more precisely measure different mechanisms of induction head behavior than in the Anthropic paper.

Causal Tracing: Like causal tracing, causal scrubbing identifies computations by patching activations. However, causal tracing aims to *identify* a specific path ("trace") that contributes causally to a particular behavior by corrupting all nodes in the neural network with noise and then iteratively denoising nodes. In contrast, causal scrubbing tries to solve a different problem: systematically *testing* hypotheses about the behavior of a whole network

by removing (“scrubbing away”) every causal relationship that should not matter according to the hypothesis being evaluated. In addition, causal tracing patches with (homoscedastic) Gaussian noise and not with the activations of other samples. Not only does this take your model off distribution, it might have no effect in cases where the scale of the activation is much larger than the scale of the noise.

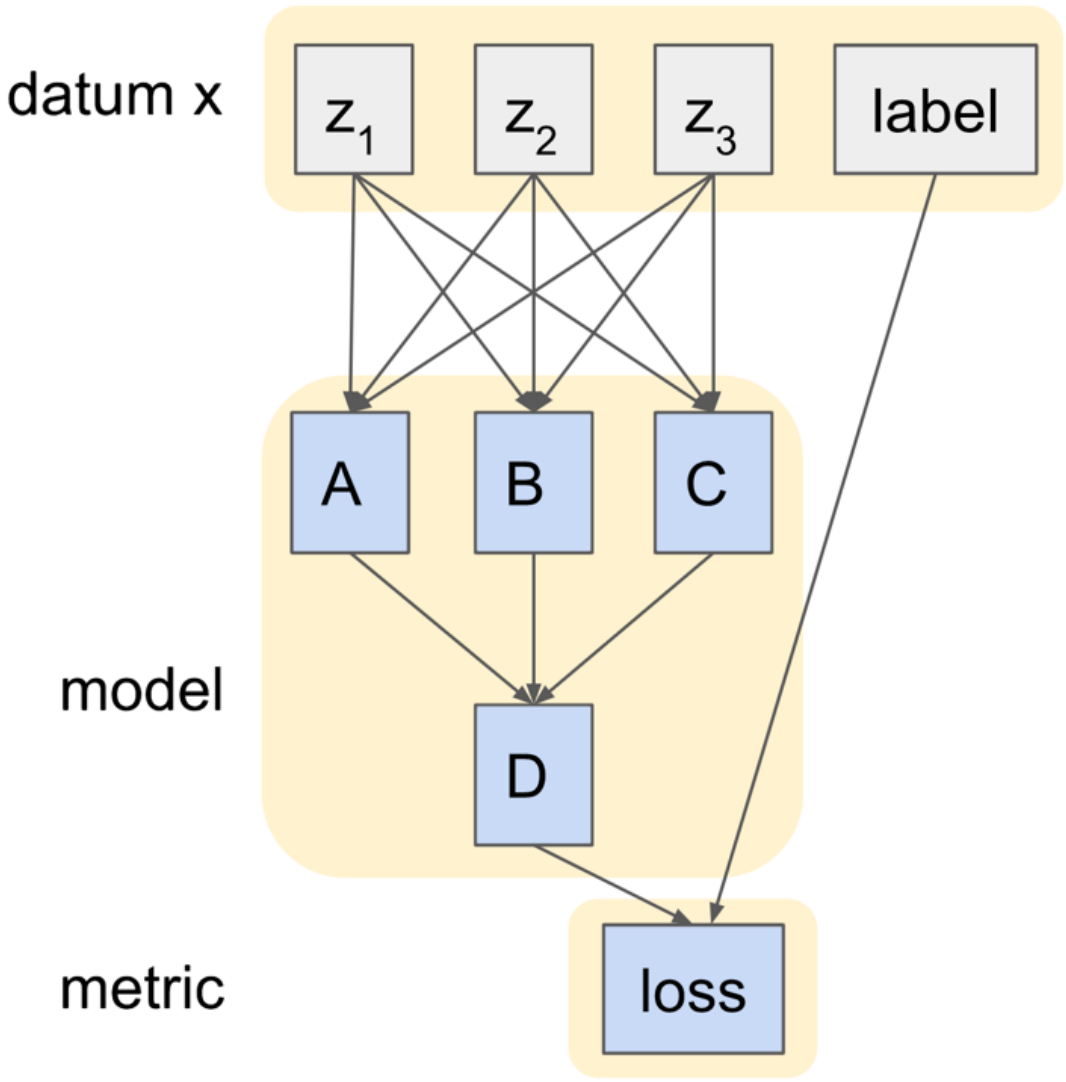
Heuristic explanations: This work takes a perspective on interpretability that is strongly influenced by ARC’s work on “heuristic explanations” of model behavior. In particular, causal scrubbing can be thought of as a form of defeasible reasoning: unlike mathematical proofs (where if you have a proof for a proposition P, you’ll never see a better proof for the negation of P that causes you to overall believe P is false), we expect that in the context of interpretability, we need to accept arguments that might be overturned by future arguments.

2 Setup

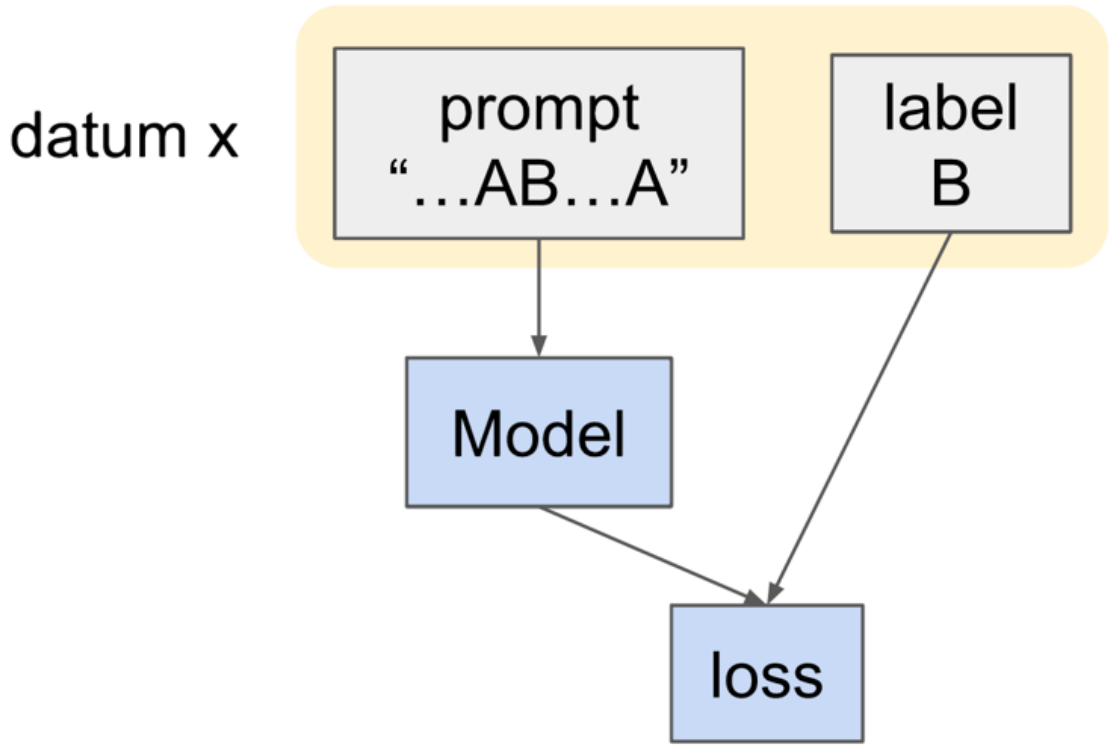
We assume a dataset D over a domain X and a function $f : X \to R$ which captures a behavior of interest. We will then explain the expectation of this function on our dataset, $E_{x \sim D}[f(x)]$.

This allows us to explain behaviors of the form “a particular model M gets low loss on a distribution D.” To represent this we include the labels in D and both the model and a loss function in f:

$$f((z, \text{label})) = \text{loss}(M(z), \text{label})$$



We also want to explain behaviors such as “if the prompt contains some bigram `AB` and ends with the token `A`, then the model is likely to predict `B` follows next.” We can do this by choosing a dataset D where each datum has the prompt `...AB...A` and expected completion `B`. For instance:

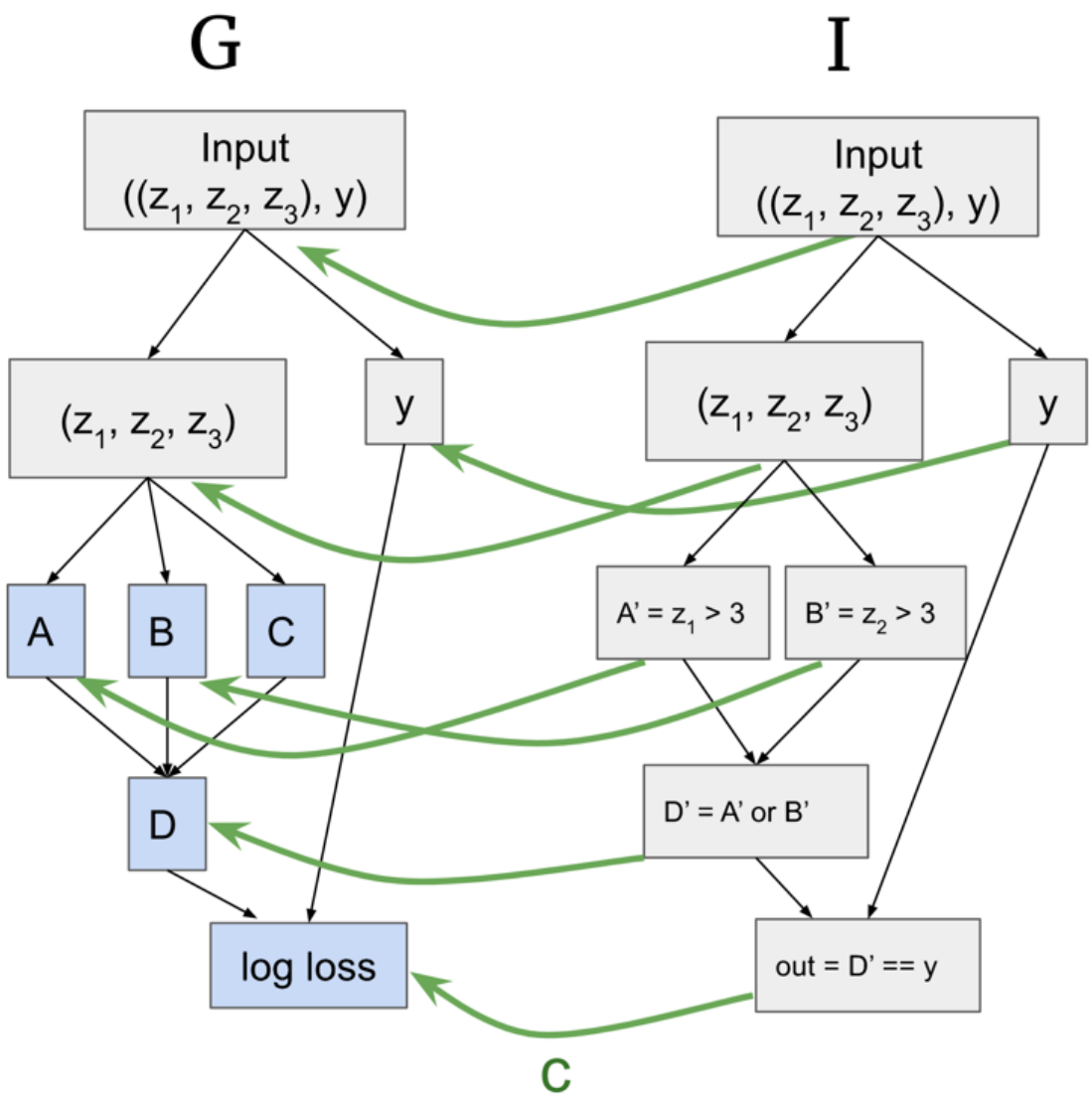


We then propose a hypothesis about how this behavior is implemented. Formally, a *hypothesis* $h = (G, I, c)$ for f is a tuple of three things:

- A computational graph G[7], which implements the function f
- We require G to be *extensionally equal* to f (equal on *all* of X)
- A computational graph I, intuitively an ‘interpretation’ of the model.
- A correspondence function c from the nodes of I to the nodes of G.
- We require c to be an injective graph homomorphism: that is, if there is an edge (u, v) in I then the edge $(c(u), c(v))$ must exist in G.

We additionally require I and G to each have a single input and output node, where c maps input to input and output to output. All input nodes are of type X which allows us to evaluate both G and I on all of X .

Here is an example hypothesis:



In this figure, we hypothesize that G works by having A compute whether $z_1 > 3$, B compute whether $z_2 > 3$, and then ORing those values. Then we're asserting that the behavior is explained by the relationship between D and the true label y.

A couple of important things to notice:

- We will often rewrite the computational graph of the original model implementation into a more convenient form (for instance splitting up a sum into terms, or grouping together several computations into one).
- You can think of I as a heuristic[8] that the hypothesis claims that the model uses to achieve the behavior. It's possible that the heuristic is imperfect and will sometimes disagree with the label y. In that case our hypothesis would claim that the model should be incorrect on these inputs.

- Note that the mapping c doesn't tell you how to translate a value of I into an activation, only which nodes correspond.
- We will call c(I) the "important nodes" of G.[9]
    - Let n_I, n_G be nodes in I and G respectively such that $c(n_I) = n_G$.
        - Intuitively this is a claim that when we evaluate both G and I on the same input, then the value of n_G (usually an activation of the model) 'represents' the value of n_I (usually a simple feature of the input).
        - The causal scrubbing algorithm will test a weaker claim: that the equivalence classes on inputs to n_I are the same as the equivalence classes on inputs to n_G. We think this is sufficient to meaningfully test the mechanistic interpretability hypotheses we are interested in, although it is not strong enough to eliminate all incorrect hypotheses.
- Among other things, the hypothesis claims that nodes of G that are not mapped to by c are unimportant for the behavior under investigation.[10]

Hypotheses are covered in more detail in the appendix.

3 Causal Scrubbing

In this section we provide two different explanations of causal scrubbing:

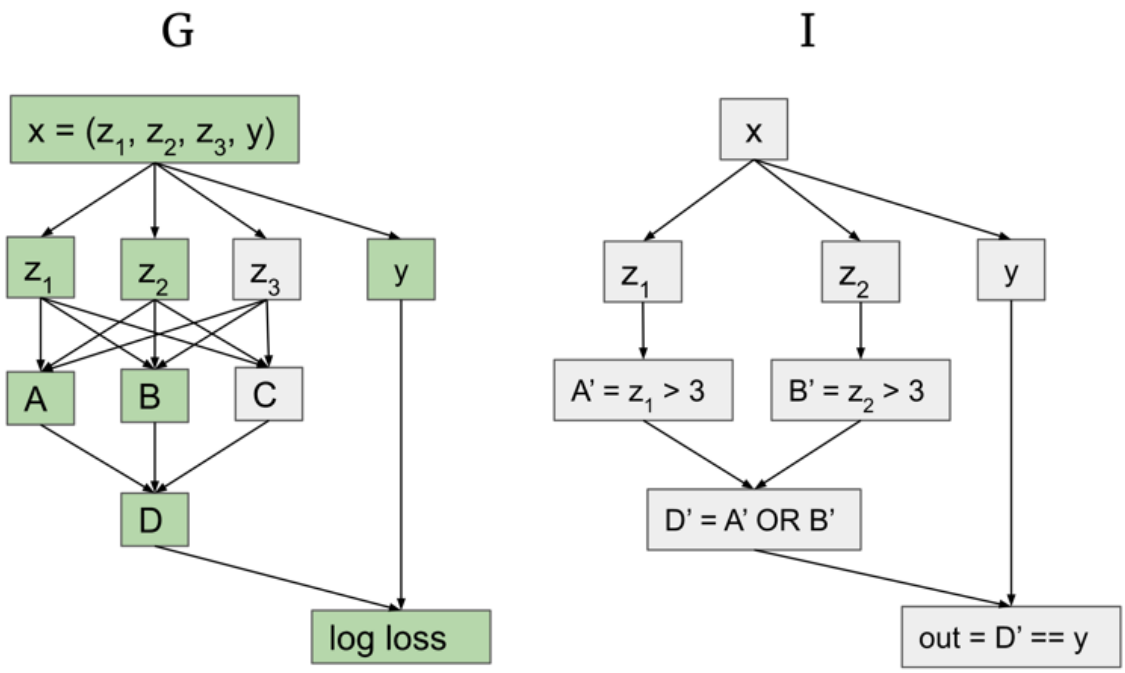
1. An informal description of the activation-replacements that a hypothesis implies are valid. We try to provide a helpful introduction to the core idea of causal scrubbing via many diagrams; and
2. The causal scrubbing algorithm and pseudocode

Different readers of this document have found different explanations to be helpful, so we encourage you to skip around or skim some sections.

Our goal will be to define a metric $E_{scrubbed}(h, D)$ by recursively sampling activations that should be equivalent according to each node of the interpretation I. We then compare this value to $E_{d\sim D}[f(d)]$. If a hypothesis is (reasonably) accurate, then the activation replacements we perform should not alter the loss and so we'd have $E_{scrubbed}(h, D) \approx E_{d\in D} f(d)$. Overall, we think that this difference will be a reasonable proxy for the *faithfulness* of the hypothesis—that is, how accurately the hypothesis corresponds to the "real reasons" behind the model behavior.[11]

3.1 An informal description: What activation replacements does a hypothesis imply are valid?

Consider a hypothesis $h = (G, I, c)$ on the graphs below, where c maps to the corresponding nodes of G highlighted in green:



This hypothesis claims that the activations A and B respectively represent checking whether the first and second component of the input is greater than 3. Then the activation D represents checking whether either of these conditions were true. Both the third component of the input and the activation of C are unimportant (at least for the behavior we are explaining, the log loss with respect to the label y).

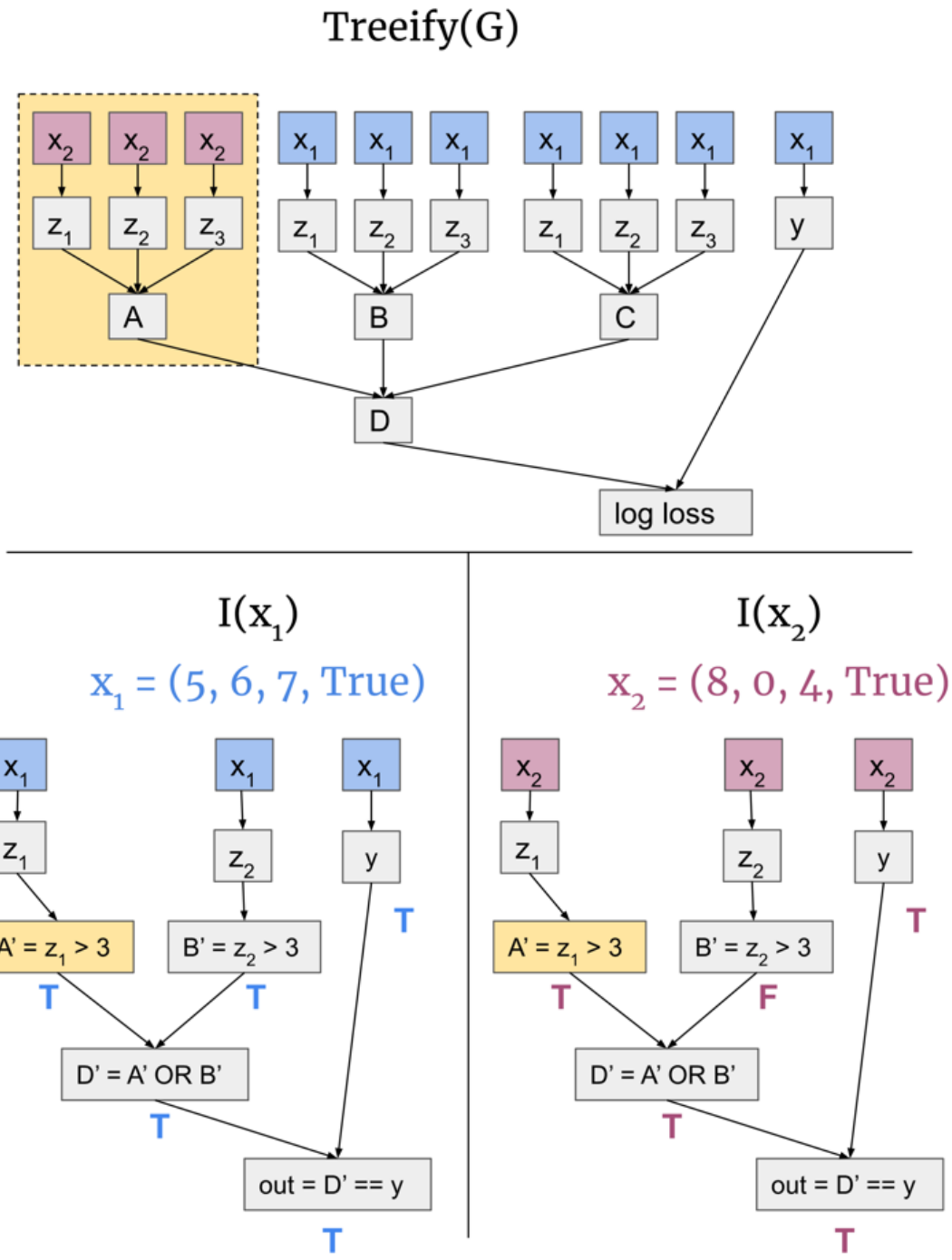
If this hypothesis is true, we should be able to perform two types of 'resampling ablations':

- replacing the activations of A, B, and D with the activations on other inputs that are "equivalent" under I; and
- replacing the activations that are claimed to be unimportant for a particular path (such as C or z_1 into B) with their activation on any other input.

To illustrate these interventions, we will depict a "treeified" version of G where every path from the input to output of G is represented by a different copy of the input. Replacing an activation with one from a different input is equivalent to replacing all inputs in the subtree upstream of that activation.

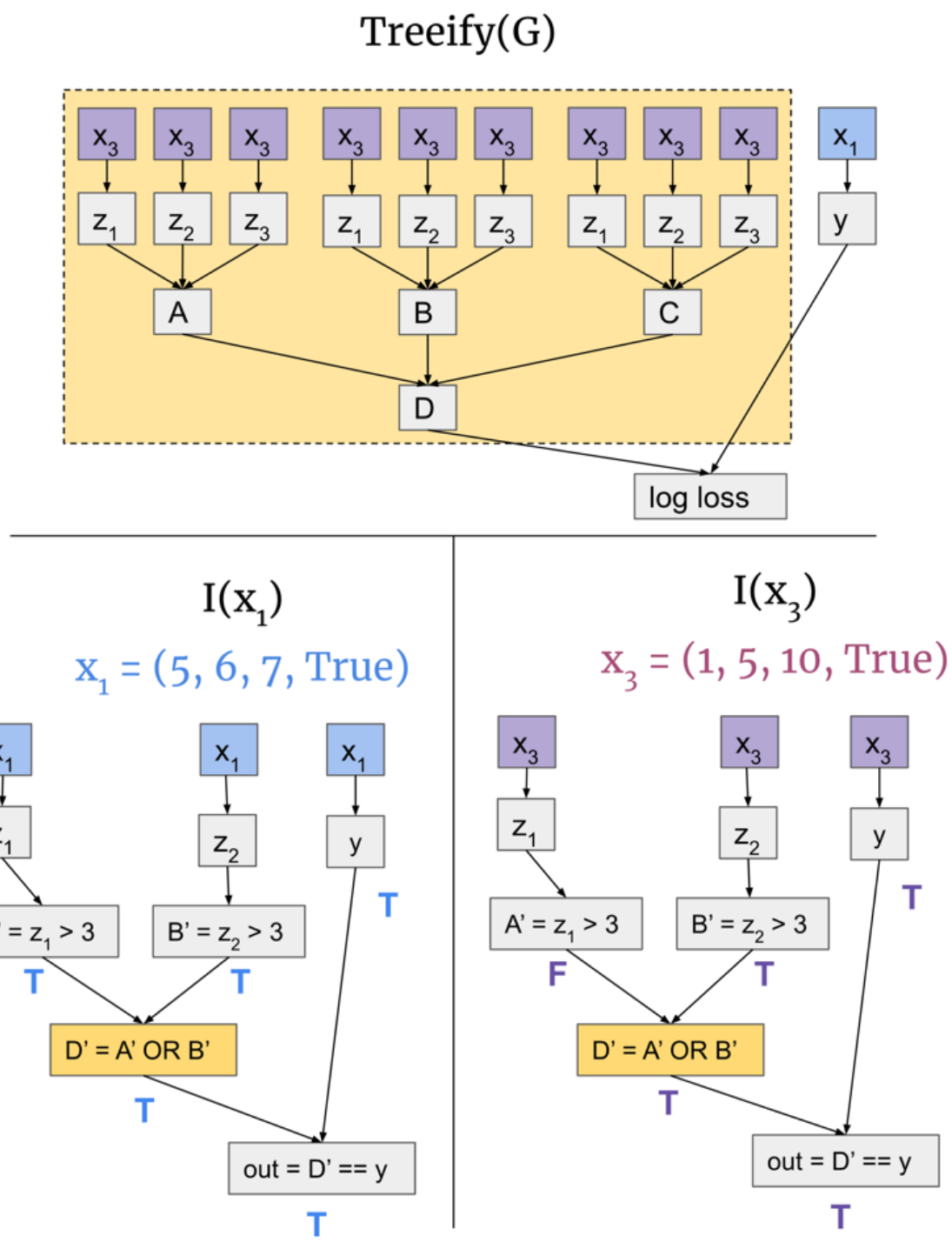
Intervention 1: semantically equivalent subtrees

Consider running the model on two inputs x_1 = (5,6,7, True) and x_2 = (8, 0, 4, True). The value of A' is the same on both x_1 and x_2. Thus, if the hypothesis depicted above is correct, the output of A on both these is equivalent. This means when evaluating G on x_1 we can replace the activation of A with its value on x_2, as depicted here:



To perform the replacement, we replaced all of the inputs upstream of A in our treeified model. (We could have performed this replacement with any other $x \in D$ that agrees on A'.)

Our hypothesis permits many other activation replacements. For example, we can perform this replacement for D instead:

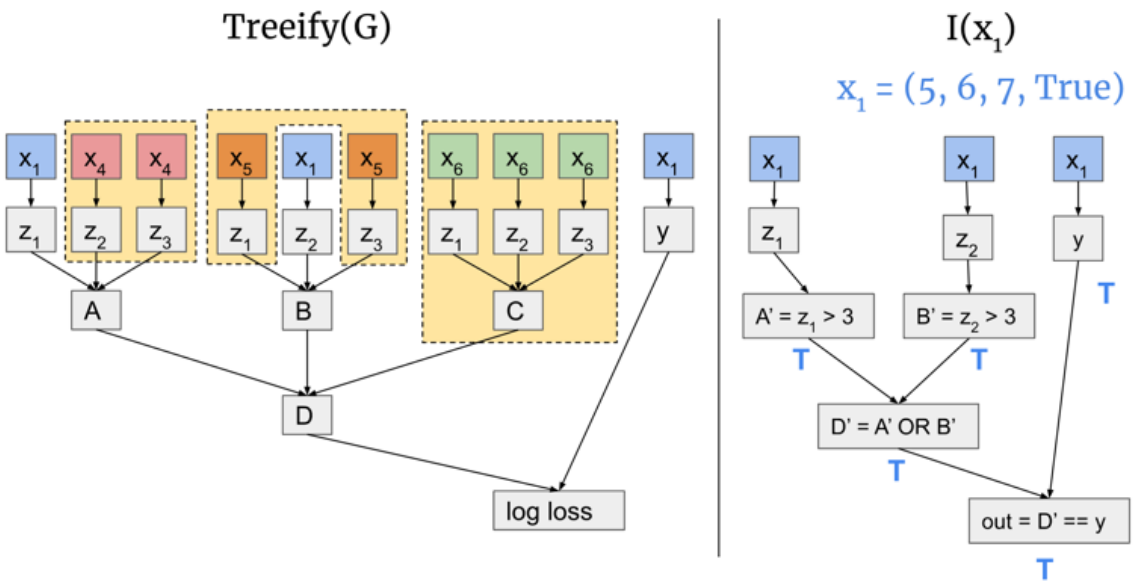


Intervention 2: unimportant inputs

The other class of intervention permitted by h is replacement of any inputs to nodes in G that h suggests aren't semantically important. For example, h says that the only important input for A is z_1. So the model's behavior should be preserved if we replace the activations

for z_2 and z_3 (or, equivalently, change the input that feeds into these activations). The same applies for z_1 and z_3 into B. Additionally, h says that D isn't influenced by C, so arbitrarily resampling all the inputs to C shouldn't impact the model's behavior.

Pictorially, this looks like this:



Notice that we are making 3 different replacements with 3 different inputs simultaneously. Still, if h is accurate, we will have preserved the important information and the output of Treeify(G)should be similar.

The causal scrubbing algorithm involves performing both of these types of intervention many times. In fact, we want to maximize the number of such interventions we perform on every run of G – to the extent permitted by h.

3.2 The causal scrubbing algorithm

We define an algorithm for evaluating hypotheses. This algorithm uses the intuition, illustrated in the previous section, of what activation replacements are permitted by a hypothesis.

The core idea is that hypotheses can be interpreted as an "intervention blacklist". We like to think of this as the hypothesis sticking its neck out and challenging us to swap around activations in any way that it hasn't specifically ruled out.

In a single sentence, the algorithm is: Whenever we need to compute an activation, we ask "What are all the other activations that, according to h, we could replace this activation with and still preserve the model's behavior?", and then make the replacement by choosing uniformly at random from that subset of the dataset, and do this recursively.

In this algorithm we don't explicitly treeify G; but we traverse it one path at a time in a tree-like fashion.

We define the ***scrubbed expectation***, $E_{scrubbed}(h, D)$, as the expectation of the behavior f over samples from this algorithm.

Intuitive Algorithm

(This is mostly redundant with the pseudocode below. Read in your preferred order.)

The algorithm is defined in pseudocode below. Intuitively we:

- Sample a random reference input x from D
- Traverse all paths through I from output towards the input by calling `run_scrub` on nodes of I recursively. For every node we consider the subgraph of I that contains everything 'upstream' of n_I (used to calculate its value from the input). Each of these correspond to a subgraph of the image c(I) in G.
- The return value of `run_scrub(n_I, c, D, x)` is an activation from G. Specifically it is an activation for the corresponding node in G that the **hypothesis claims represents the value of** n_I when I is run on input `x`.
    - Let $n_G = c(n_I)$.
    - If n_G is an input node we will return x.
    - Otherwise we will determine the activations of each input from the parents of n_G. For each parent p_G of n_G:
- If there exists a parent p_I of n_I that corresponds to p_G then the hypothesis claims that the value of p_G is important for n_G. In particular it is important as it represents the value defined by p_I. Thus we sample a datum `new_x` that agrees with x on the value of p_I. We'll **recursively call** `run_scrub` on p_I in order to get an activation for p_G.
- For any "unimportant parent" not mapped by the correspondence, we select an input `other_x`. This is a random input from the dataset, however we enforce that the *same* random input is used by all unimportant parents of a particular node.[12] We record the value of p_G on `other_x`.
- We now have the activations of all the parents of n_G – these are exactly the inputs to running the function defined for the node n_G. We return the output of this function.

Pseudocode

```
        def estim(h, D):
"""Estimate E_scrubbed(h, D)"""
_G, I, c = h
outs = []
for i in NUM_SAMPLES:
    x = random.sample(D)
    outs.append(run_scrub(c, D, output_node_of(I), x))
return mean(outs)
```

```
def run_scrub(
    c,  # correspondence I -> G
    D: Set[Datum],
    n_I, # node of I
    ref_x: Datum
):
    """Returns an activation of n_G which h claims represents n_I(ref_x)."""
    n_G = c(n_I)

    if n_G is an input node:
        return ref_x

    inputs_G = {}

    # pick a random datum to use for all “unimportant parents” of this node
    random_x = random.sample(D)

    # get the scrubbed activations of the inputs to n_G
    for parent_G in n_G.parents():
        # “important” parents
        if parent_G is in map(c, n_I.parents()):
            parent_I = c.inverse(parent_G)
            # sample a new datum that agrees on the interpretation node
            new_x = sample_agreeing_x(D, parent_I, ref_x)
            # and get its scrubbed activations recursively
            inputs_G[parent_G] = run_scrub(c, D, parent_I, new_x)
        # “unimportant” parents
        else:
            # get the activations on the random input value chosen above
            inputs_G[parent_G] = parent_G.value_on(random_x)

    # now run n_G given the computed input activations
    return n_G.value_from_inputs(inputs_G)

def sample_agreeing_x(D, n_I, ref_x):
    """Returns a random element of D that agrees with ref_x on the value of n_I"""
    D_agree = [x in D if n_I.value_on(ref_x) == n_I.value_on(x)]
    return random.sample(D_agree)
```

4 Why ablate by resampling?

4.1 What does it mean to say “this thing doesn’t matter”?

Suppose a hypothesis claims that some module in the model isn’t important for a given behavior. There are a variety of different interventions that people do to test this. For example:

- Zero ablation: setting the activations of that module to 0
- Mean ablation: replacing the activations of that module with their empirical mean on D
- Resampling ablation: patching in the activation of that module on a random different input

In order to decide between these, we should think about the precise claim we’re trying to test by ablating the module.

If the claim is “this module’s activations are literally unused”, then we could try replacing them with huge numbers or even NaN. But in actual cases, this would destroy the model

behavior, and so this isn't the claim we're trying to test.

We think a better type of claim is: "The behavior might depend on various properties of the activations of this module, but those activations aren't encoding any information that's relevant to this subtask." Phrased differently: The distribution of activations of this module is (maybe) important for the behavior. But we don't depend on any properties of this distribution that are conditional on *which* particular input the model receives.

This is why, in our opinion, the most direct way to translate this hypothesis into an intervention experiment is to patch in the module's activation on a randomly sampled different input–this distribution will have all the properties that the module's activations usually have, but any connection between those properties and the correct prediction will have been scrubbed away.

4.2 Problems with zero and mean ablation

Despite their prevalence in prior work, zero and mean ablations do not translate the claims we'd like to make faithfully.

As noted above, the claim we're trying to evaluate is that the information in the output of this component doesn't matter for our current model, not the claim that deleting the component would have no effect on behavior. We care about evaluating the claim as faithfully as possible on our current model and not replacing it with a slightly different model, which zero or mean ablation of a component does. This core problem can manifest in three ways:

1. *Zero and mean ablations take your model off distribution in an unprincipled manner.*
2. *Zero and mean ablations can have unpredictable effects on measured performance.*
3. *Zero and mean ablations remove variation and thus present an inaccurate view of what's happening.*

For more detail on these specific issues, we refer readers to the appendix post.

5 Results

To show the value of this approach, we apply causal scrubbing algorithm to two tasks: 1) verifying hypotheses about an algorithmic model we found previously through ad-hoc interpretability, and 2) test and incrementally improve hypotheses about how induction heads work on a 2-layer attention only model. Here, we summarize the results of those applications here to illustrate the applications of causal scrubbing; detailed results can be found in the respective auxiliary posts.

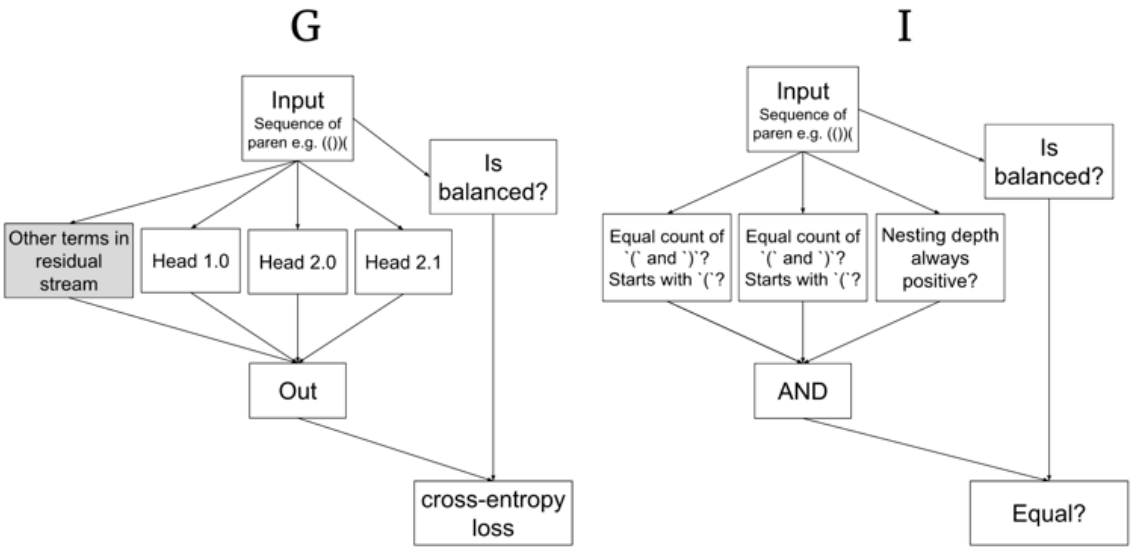
5.1 On a paren balance checker

We apply the causal scrubbing algorithm to a small transformer which classifies sequences of parentheses as balanced or unbalanced; see the results post for more information. In particular, we test three claims about the mechanisms this model uses.

Claim 1: There are three heads that directly pass important information to output:[13]

- Heads 1.0 and 2.0 test the conjunction of two checks: that there are an equal number of open and close parentheses in the entire sequence, and that the sequence starts open.
- Head 2.1 checks that the nesting depth is never negative at any point in the sequence.

Claim 1 is represented by the following hypothesis:[14]



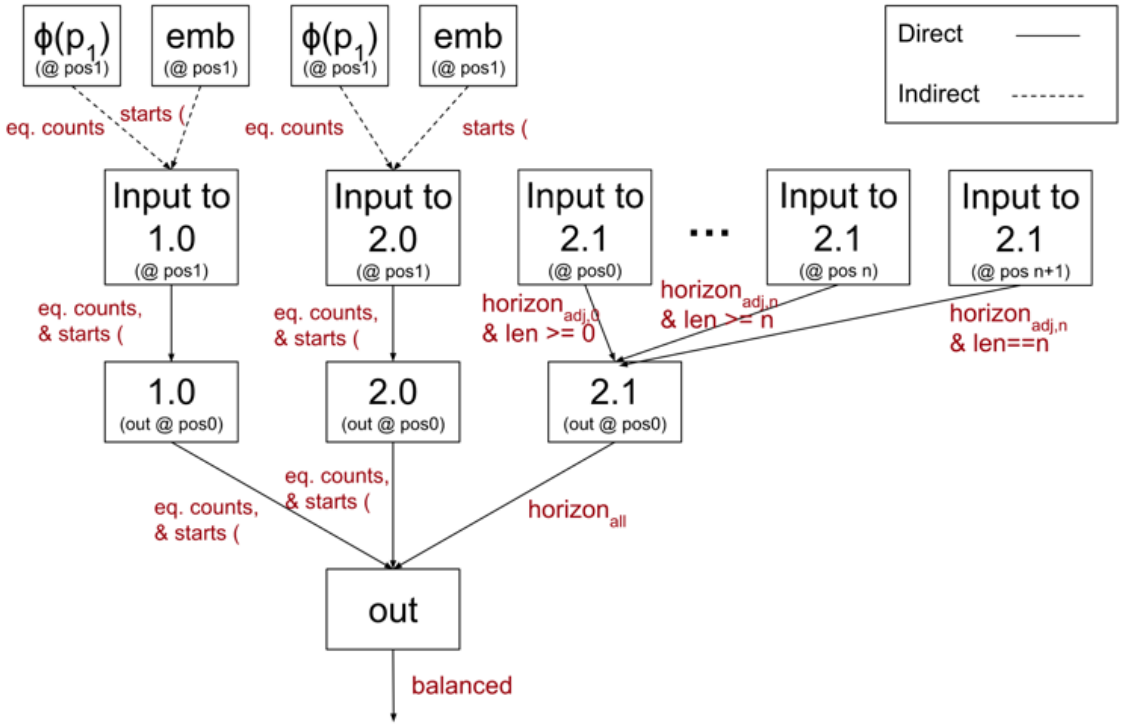
The hypothesis for claim 1. The correspondence in this diagram maps to all the nodes of G except the "other terms" node in gray. The "is balanced?" node in both graphs algorithmically computes if the input is balanced with perfect accuracy in order to compute the loss for the model. The node labeled "Equal count of `(` and `)`? Starts with `(`?" computes the conjunction of both these two checks.

Claim 2: Heads 1.0 and 2.0 depend only on their input at position 1, and this input indirectly depends on:

1. The output of 0.0 at position 1, which computes the overall proportion of parentheses which are open. This is written into a particular direction of the residual stream in a linear fashion.
2. The embedding at position 1, which indicates if the sequence starts with `(`.

Claim 3: Head 2.1 depends on the input at all positions, and if the nesting depth (when reading right to left!) is negative at that position.[15]

Here is a visual representation of the combination of all three claims:



A representation of the hypothesis for all three claims. Arrows are annotated with the feature of the interpretation corresponding to the parent node. Inputs claimed to be unimportant not shown. ϕ is a function from [0,1] to the embedding space that we claim represents the important part of the output of head 0.0 (the residual between the actual output of 0.0 and this estimate is thus claimed to be unimportant and we perform a replacement ablation on).

Testing these claims with causal scrubbing, we find that they are reasonably, but not completely, accurate:

Claim(s) tested	Performance recovered[16]
1	93%
1 + 2	88%
1 + 3	84%
1 + 2 + 3	72%

As expected, performance drops as we are more specific about how exactly the high level features are computed. This is because as the hypotheses get more specific, they induce more activation replacements, often stacked several layers deep.[17]

This indicates our hypothesis is subtly incorrect in several ways, either by missing pathways along which information travels or imperfectly identifying the features that the model uses in practice.

We explain these results in more detail in this appendix post.

5.2 On induction

We investigated ‘induction’ heads in a 2 layer attention only model. We were able to easily test out and incrementally improve hypotheses about which computations in the model were important for the behavior of the heads.

We first tested a naive induction hypothesis, which separates out the input to an induction head in layer 1 into three separate paths – the value, the key, and the query – and specified where the important information in each path comes from. We hypothesized that both the values and queries are formed based on only the input directly from the token embeddings via the residual stream and have no dependence on attention layer 0. The keys, however, are produced only by the input from attention layer 0; in particular, they depend on the part of the output of attention layer 0 that corresponds to attention on the previous token position.[18]

We test these hypotheses on a subset of openwebtext where induction is likely (but not guaranteed) to be helpful.[19] Evaluated on this dataset, this naive hypothesis only recovers 35% of the performance. In order to improve this we made various edits which allow the information to flow through additional pathways:

- First, we allow the attention pattern of the induction head to compare a set of three consecutive tokens (instead of just a single token) to determine when to induct.
- Next, we also allow the query and value to also depend on the part of the output of layer 0 that corresponds to the current position.
- We also special case three layer 0 heads which attend to repeated occurrences of the current token. In particular, we assume that the important part of the output of these heads is what their output would be *if* their attention was just an identity matrix.[20]

With these adjustments, our hypothesis recovers 86% of the performance.

We believe it would have been significantly harder to develop and have confidence in a hypothesis this precise only using ad-hoc methods to verify the correctness of a hypothesis.

We explain these results in more detail in this appendix post.

6 Relevance to alignment

The most obvious application of causal scrubbing to alignment is using it to evaluate mechanistic interpretations. In particular, we can imagine several specific use cases that are relevant to alignment:

- *Checking interpretations of model behaviors produced by human researchers.* Having a standardized, reliable, and convenient set of tests would make it much easier to scale up mechanistic interpretability efforts; this might be particularly important if there are big interpretability projects right before the deployment of transformative AI.
- *Automated algorithmic searches for explanations.* In some cases, researchers might be able to specify a space of hypotheses and then use optimization algorithms to find the most predictive ones. We’ve done some work like this and we hope to do much more in the future.
- *AI-assisted explanations.* We might be able to train models to produce highly rated and human-understandable explanations.

In all three applications, we required that researchers understand the explanations that were verified by causal scrubbing. Unfortunately, it might be the case that the behaviors we want to interpret in large neural networks won’t have *any* understandable interpretations at all if most of the cognition performed inside powerful AI systems is in some sense irreducibly complex. It also seems plausible that even if these human-understandable interpretations exist, it might be intractable or impractical to find them.

A lot of our interest in causal scrubbing (and mechanistic interpretability more generally) comes from applications which require interpretability-like techniques which rely on formally manipulating explanation-like objects but *don't* require that these objects be understood by anyone (human or AI):

- *Automated strategies for solving ELK.* ARC is optimistic about some strategies for solving ELK that involve searching for objects similar to causal scrubbing explanations and then using properties of these explanations as part of the training procedure of the model, in ways that don't require humans to understand the explanations.
- *Detecting deceptive alignment.* Suppose you have a weak trusted model and a strong untrusted model. You might be able to search for explanations of why these models take similar actions which allow you to distinguish whether the untrusted model is deceptively aligned just based on the structure of the explanation, rather than via having to understand its content.
- *Relaxed adversarial training* requires some way of adjudicating arguments about whether the internals of models imply they'll behave badly in ways that are hard to find with random sampling (because the failures only occur off the training distribution, or they're very rare). This doesn't require that any human is able to understand these arguments; it just requires we have a mechanical argument evaluation procedure. Improved versions of the causal scrubbing algorithm might be able to fill this gap.

7 Limitations

Unfortunately, causal scrubbing may not be able to express all the tests of interpretability hypotheses we might want to express:

- Causal scrubbing only allows activation replacements that are *perfectly permissible* by the hypothesis: that is, the respective inputs have an exactly equal value in the correspondance.
    - Despite being maximally strict in what replacements to allow, we are in practice willing to accept hypotheses that fail to perfectly preserve performance. We think this is an inconsistency in our current approach.
    - As a concrete example, if you think a component of your model encodes a continuous feature, you might want to test this by replacing the activation of this component with the activation on an input that is *approximately* equal on this feature–causal scrubbing will refuse to do this swap.
    - You can solve this problem by considering a generalized form of causal scrubbing, where hypotheses specify a non-uniform distribution over swaps. We've worked with this "generalized causal scrubbing" algorithm a bit. The space of hypotheses is continuous, which is nice for a lot of reasons (e.g. you can search over the hypothesis space with SGD). However, there are a variety of conceptual problems that still need to be resolved (e.g. there are a few different options for defining the union of two hypotheses, and it's not obvious which is most principled).
- Causal scrubbing can only propose tests that can be constructed using the data provided to it. If your hypothesis predicts that model performance will be preserved if you swap the input to any other input which has a particular property, but no other inputs in the dataset have that property, causal scrubbing can't test your hypothesis. This happens in practice–there is probably only one sequence in webtext with a particular first name at token positions 12, 45, and 317, and a particular last name at 13, 46, 234.
    - This problem is addressed if you are able to produce samples that match properties by some mechanism other than rejection sampling.
- Causal scrubbing doesn't allow us to distinguish between two features that are perfectly correlated on our dataset, since they would induce the same equivalence classes. In fact, to the extent that two features A and B are highly correlated, causal scrubbing will not complain if you misidentify an A-detector as a B-detector.[21]

Another limitation is that causal scrubbing does not guarantee that it will reject a hypothesis that is importantly false or incomplete. Here are two concrete cases where this happens:

- When a model uses some heuristic that isn't *always* applicable, it might use other circuits to inhibit the heuristic (for example, the negative name mover heads in the Indirect Object Identification paper). However, these inhibitory circuits are purely harmful for inputs where the heuristic *is* applicable. In these cases, if you ignore the inhibitory circuits, you might overestimate the contribution of the heuristic to performance, leading you to falsely believe that your incomplete interpretation fully explains the behavior (and therefore fail to notice other components of the network that contribute to performance).
- If two terms are correlated, sampling them independently (by two different random activation swaps) reduces the variance of the sum. Sometimes, this variance can be harmful for model performance – for instance, if it represents interference from polysemanticity. This can cause a hypothesis that scrubs out correlations present in the model's activations to appear 'more accurate' under causal scrubbing.[22]

These examples are both due to the hypotheses not being specific *enough* and neglecting to include some correlation in the model (either between input-feature and activation or between two activations) that would hurt the performance of the scrubbed model.

We don't think that this is a problem with causal scrubbing in particular; but instead is because interpretability explanations should be regarded as an example of defeasible reasoning, where it is possible for an argument to be overturned by further arguments.

We think these problems are fairly likely to be solvable using an adversarial process where hypotheses are tested by allowing an adversary to modify the hypothesis to make it more specific in whatever ways affect the scrubbed behavior the most. Intuitively, this adversarial process requires that proposed hypotheses "point out all the mechanisms that are going on that matter for the behavior", because if the proposed hypothesis doesn't point something important out, the adversary can point it out. More details on this approach are included in the appendix post.

Despite these limitations, we are still excited about causal scrubbing. We've been able to directly apply it to understanding the behaviors of simple models and are optimistic about it being scalable to larger models and more complex behaviors (insofar as mechanistic interpretability can be applied to such problems at all). We currently expect causal scrubbing to be a big part of the methodology we use when doing mechanistic interpretability work in the future.

Acknowledgements

This work was done by the Redwood Research interpretability team. We're especially thankful for Tao Lin for writing the software that we used for this research and for Kshitij Sachan for contributing to early versions of causal scrubbing. Causal scrubbing was strongly inspired by Kevin Wang, Arthur Conmy, and Alexandre Variengien's work on how GPT-2 Implements Indirect Object Identification . We'd also like to thank Paul Christiano and Mark Xu for their insights on heuristic arguments on neural networks. Finally, thanks to Ben Toner, Oliver Habryka, Ajeya Cotra, Vladimir Mikulik, Tristan Hume, Jacob Steinhardt, Neel Nanda, Stephen Casper, and many others for their feedback on this work and prior drafts of this sequence.

1. ^

   For example, in the causal tracing paper (Meng et al 2022), to evaluate whether their hypothesis correctly identified the location of facts in GPT-2, the authors replace the activation of the involved neurons and observed that the model behaved as though it

believed the edited fact, and not the original fact. In the Induction Heads paper (Olsson et al 2022) the authors provide six different lines of evidence, from macroscopic co-occurrence to mechanistic plausibility.

2. ^

   Causal scrubbing is technically formulated in terms of general computational graphs, but we're primarily interested in using causal scrubbing on computational graphs that implement neural networks.

3. ^

   See the discussion in the "An alternative formalism: constructing a distribution on treeified inputs" section of the appendix post.

4. ^

   Most commonly, the behavior we attempt to explain is why a model achieves low loss on a particular set of examples, and thus we measure the loss directly. However, the method can explain any expected quality of the model's output.

5. ^

   We expect the results posts will be especially useful for people who wish to apply causal scrubbing in their own research.

6. ^

   Note that we can use causal scrubbing to ablate a particular module, by using a hypothesis where that specific module's outputs do not matter for the model's performance.

7. ^

   A computational graph is a graph where the nodes represent computations and the edges specify the inputs to the computations.

8. ^

   In the normal sense of the word, not ARC's Heuristic Arguments approach

9. ^

   Since c is required to be an injective graph homomorphism, it immediately follows that c(I) is a subgraph of G which is isomorphic to I. This subgraph will be a union of paths from the input to the output.

10. ^

    In the appendix we'll discuss that it is possible to modify the correspondence to include these unimportant nodes, and that doing so removes some ambiguity on when to sample unimportant nodes together or separately.

11. ^

    We have no guarantee, however, that any hypothesis that passes the causal scrubbing test is desirable. See more discussion of counterexamples in the limitations section.

12. ^

    This is because otherwise our algorithm would crucially depend on the exact representation of the causal graph: e.g. if the output of a particular attention layer was represented as a single input or if there was one input per attention head instead. There are several other approaches that can be taken to addressing this ambiguity, see the appendix.

13. ^

    That is, we consider the contribution of these heads through the residual stream into the final layer norm, excluding influence they may have through intermediate layers.

14. ^

    Note that as part of this hypothesis we have aggressively simplified the original model into a computational graph with only 5 separate computations. In particular, we relied on the fact that residual stream just before the classifier head can be written as a sum of terms, including a term for each attention head (see "Attention Heads are Independent and Additive" section of Anthropic's "Mathematical Framework for Transformer Circuits" paper). Since we claim only three of these terms are important, we clump all other terms together into one node. Additionally note this means that the 'Head 2.0' node in G includes *all* of the computations from layers 0 and 1, as these are required to compute the output of head 2.0 from the input.

15. ^

    The claim we test is somewhat more subtle, involving a weighted average between the proportion of the open-parentheses in the prefix and suffix of the string when split at every position. This is equivalent for the final computation of balancedness, but more closely matches the model's internal computation.

16. ^

    As measured by normalizing the loss so 100% is loss of the normal model (0.0003) and 0% is the loss when randomly permuting the labels. For the reasoning behind this metric see the appendix.

17. ^

    Our final hypothesis combines up to 51 different inputs: 4 inputs feeding into each of 1.0 and 2.0, 42 feeding into 2.1 (one for each sequence position), and 1 for the 'other terms'.

18. ^

    The output of an attention layer can be written as a sum of terms, one for each previous sequence position. We can thus claim that only one of these terms is important for forming the queries.

19. ^

    In particular we create a whitelist of tokens on which exact 2-token induction is often a helpful heuristic (over and above bigram-heuristics). We then filter openwebtext (prompt, next-token) pairs for prompts that end in tokens on our whitelist. We evaluate loss on the actual next token from the dataset, however, which may not be what induction expects. More details here.

We do this as we want to understand not just how our model implements induction but also how it decides *when* to use induction.

20. ^

And thus the residual of (actual output - estimated output) is unimportant and can be interchanged with the residual on any other input.

21. ^

This is a common way for interpretability hypotheses to be 'partially correct.' Depending on the type of reliability needed, this can be more or less problematic.

22. ^

Another real world example of this is this this experiment on the paren balance checker

Causal scrubbing: Appendix

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

* *Authors sorted alphabetically.*

An appendix to this post .

1 More on Hypotheses

1.1 Example behaviors

As mentioned above, our method allows us to explain quantitatively measured model behavior operationalized as the expectation of a function f on a distribution D.

Note that no part of our method distinguishes between the part of the input or computational graph that belongs to the "model" vs the "metric."[1]

It turns out that you can phrase a lot of mechanistic interpretability in this way. For example, here are some results obtained from attempting to explain how a model has low loss:

- Nanda and Lieberum's analysis of the structure of a model that does modular addition explains the observation that their model gets low loss on the validation dataset.
- The indirect object identification circuit explains the observation that the model gets low loss on the indirect object identification task, as measured on a synthetic distribution.
- Induction circuits (as described in Elhage et al. 2021) explain the observation that the model gets low loss when predicting tokens that follow the heuristic: "if AB has occurred before, then A is likely to be followed by B".

That being said, you can set up experiments using other metrics besides loss as well:

- Cammarata et al identify curve detectors in the Inception vision model by using the response of various filters on synthetic datasets to explain the correlation between: 1) the activation strength of some neuron, and 2) whether the orientation of an input curve is close to a reference angle.

1.2 Extensional equality, and common rewrites of G and I

If you're trying to explain the expectation of f, we always consider it a valid move to suggest an alternative function f' if $f(x) = f'(x)$ on every input ("extensional equality"), and then explain f' instead. In particular, we'll often start with our model's computational graph and a simple interpretation, and then perform "algebraic rewrites" on both graphs to naturally specify the correspondence.

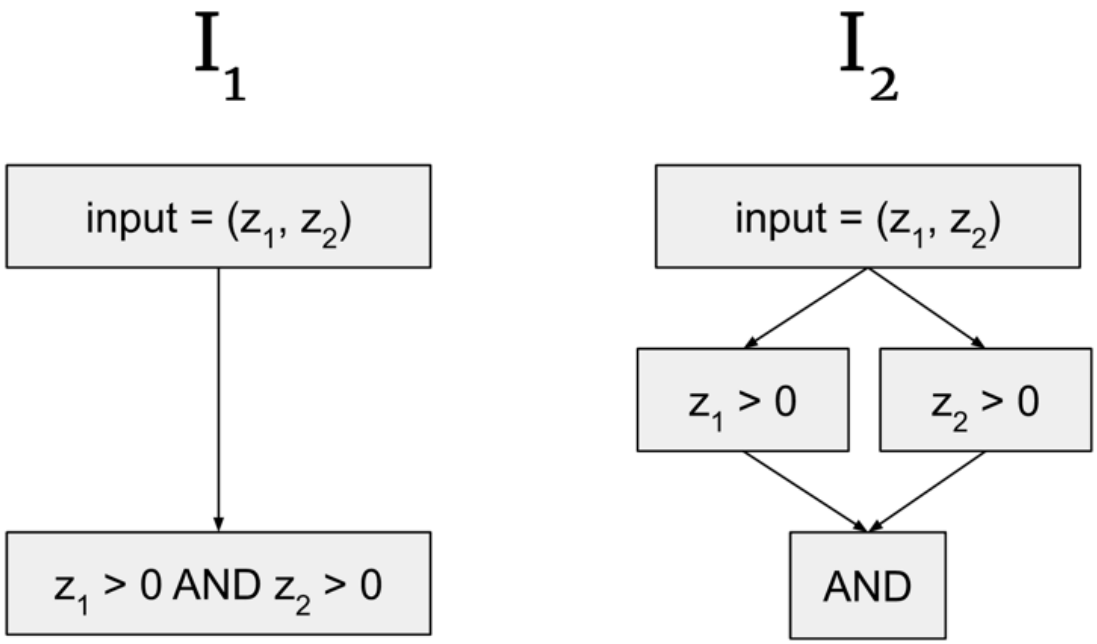
Common rewrites include:

- When the output of a single component of the model is used in different ways by different paths, we'll duplicate that node in G, such that each copy can correspond to a different part of I.
- When multiple components of the model compute a single feature we can either:
- duplicate the node in I, to sample the components *separately*; or
- combine the nodes of G into a single node, to sample the components *together.*
- Sometimes, we want to test claims of the form "this subspace of the activation contains the feature of interest". We can express this by rewriting the output as a sum of the activation projected into subspace and the orthogonal component. We can then propose that only the projected subspace encodes the feature.
- An even more complicated example is when we want to test a theorized function ϕ that maps from an input to a predicted activation of a component. We can then rewrite the output as the sum of two terms: ϕ(input) and the residual (the error of the estimate), and then claim only the phi term contains important information. If your estimate is bad, the error term will be large in important ways. This is especially useful to test hypotheses about scalar quantities (instead of categorical ones).[2]

Note that there are many trivial or unenlightening algebraic rewrites. For example, you could always replace f' with a lookup table of f, and in cases where the model performs perfectly, you can also replace f with the constant zero function. Causal scrubbing is *not* intended to generate mechanistic interpretations or ensure that only mechanistic interpretations are allowed, but instead to check that a given interpretation is faithful. We discuss this more in the limitations section of the main post.

1.3 Interpretations at multiple levels of specificity

We allow hypotheses at a wide variety of levels of specificity. For example, here are two potential interpretations of the same f:



These interpretations correspond to the same input-output mappings, but the hypothesis on the right is more specific, because it's saying that there are three separate nodes in the graph expressing this computation instead of one. So when we construct G_2 to correspond to I_2 we would need three different activations that we claim are important in different ways, instead of just one for G_1 mapping to I_1. In interpretability, we all-else-equal prefer more specific explanations, but defining that is out of scope here–we're just trying to provide a way of looking at the predictions made by hypotheses, rather than expressing any a priori preference over them.

2 What metric should causal scrubbing use?

2.1 "Percentage of loss recovered" as a measure of hypothesis quality

In both of these results posts, in order to measure the similarity between the scrubbed and unscrubbed models, we use *% loss recovered*.

As a baseline we use $E_{randomized}$, the 'randomized loss', defined as the loss when we shuffle the connection between the correct labels and the model's output. Note this randomized loss will be higher than the loss for a calibrated guess with no information. We use randomized loss as the baseline since we are interested in explaining why the model makes the guesses it makes. If we had no idea, we could propose the trivial correspondence that the model's inputs and outputs are unrelated, for which $E_{scrubbed} = E_{randomized}$.

Thus we define:

$$\%\ \text{loss recovered}(E_{\text{model}}, E_{\text{scrubbed}}) = \frac{E_{\text{scrubbed}} - E_{\text{randomized}}}{E_{\text{model}} - E_{\text{randomized}}} 100\%.$$

This percentage can exceed 100% or be negative. It is not very meaningful as a fraction, and is rather an arithmetic aid for comparing the magnitude of expected losses under various distributions. However, it is the case that hypotheses with a "% loss recovered" closer to 100% result in predictions that are more consistent with the model.

2.2 Why not compare the full distribution, rather than expectations?

Above, we rate our hypotheses using the distance between the expectation under the dataset and the scrubbed distribution, $|E[f(x)] - E_{\text{scrubbed}}(h, D)|$.[3]

You could instead rate hypotheses by comparing the full distribution of input-output behavior. That is, the difference between the distribution of the random variable f(x) under the data set D, and f(x) under D_{scrubbed}.

In this work, we prefer the expected loss. Suppose that one of the drivers of the model's behavior is noise: trying to capture the full distribution would require us to explain what causes the noise. For example, you'd have to explain the behavior of a randomly initialized model despite the model doing 'nothing interesting'.

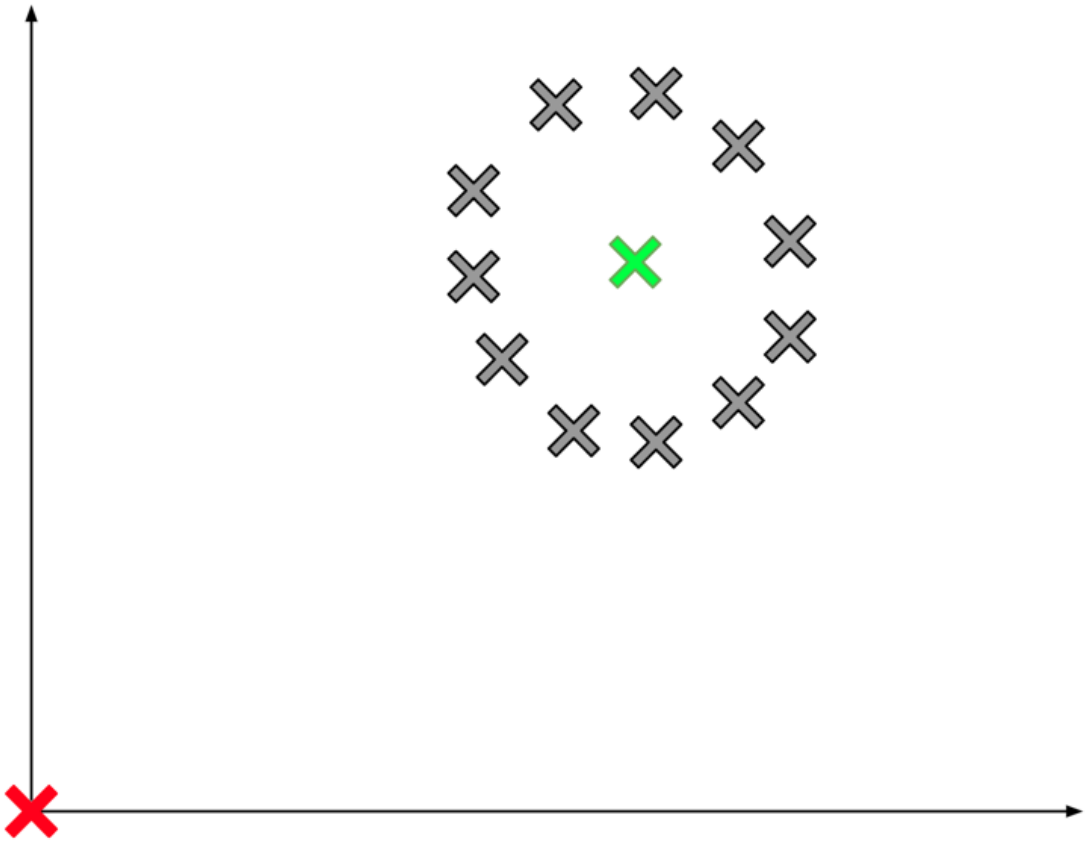
3 Further discussion of zero and mean ablation

Earlier, we noted our preference for "resampling ablation" of a component of a model (patch an activation of that component from a randomly selected input in the dataset) over zero or mean ablation of that component (set that component's activation to 0 or its mean over the entire dataset, respectively) in order to test the claim "this component doesn't matter for our explanation of the model". We also mentioned three specific problems we see with using zero or mean ablation to test this claim. Here, we'll discuss these problems in greater detail.

1) Zero and mean ablations take your model off distribution in an unprincipled manner.

The first problem we see with these ablations is that they destroy various properties of the distribution of activations in a way that seems unprincipled and could lead to the ablated model performing either worse or better than it should.

As an informal argument, imagine we have a module whose activations are in a two dimensional space. In the picture below we've drawn some of its activations as gray crosses, the mean as a green cross, and the zero as a red cross:



It seems to us that zero ablating takes your model out of distribution in an unprincipled way. (If the model was trained with dropout, it's slightly more reasonable, but it's rarely clear how a model actually handles dropout internally.) Mean ablating also takes the model out of distribution because the mean is not necessarily on the manifold of plausible activations.

2) Zero and mean ablations can have unpredictable effects on measured performance.

Another problem is that these ablations can have unpredictable effects on measured performance. For example, suppose that you're looking at a regression model that happens to output larger answers when the activation from this module is at its mean activation (which, let's suppose, is off-distribution and therefore unconstrained by SGD). Also, suppose you're looking at it on a data distribution where this module is in fact unimportant. If you're analyzing model performance on a data subdistribution where the model generally guesses too high, then mean ablation will make it look like ablating this module harms performance. If the model generally guesses too low on the subdistribution, mean ablation will improve performance. Both of these failure modes are avoided by using random patches, as resampling ablation does, instead of mean ablation.

3) Zero and mean ablations remove variation that your model might depend on for performance.

The final problem we see with these ablations is that they neglect the variation in the outputs of the module. Removing this variation doesn't seem reasonable when claiming that the module doesn't matter.

For an illustrative toy example, suppose we're trying to explain the performance of a model with three modules M1, M2, and M3. This model has been trained with dropout and usually only depends on components M1 and M2 to compute its output, but if dropout is active and knocks out M2, the model uses M3 instead and can perform almost as well as if it were able to use M1 and M2.

If we zero/mean ablate M2 (assume mean 0), it will look like M2 wasn't doing anything at all and our hypothesis that it wasn't relevant will be seemingly vindicated. If instead we resample ablate M2, the model will perform significantly worse (exactly how much worse is dependent on exactly how the output of M2 is relevant to the final output).

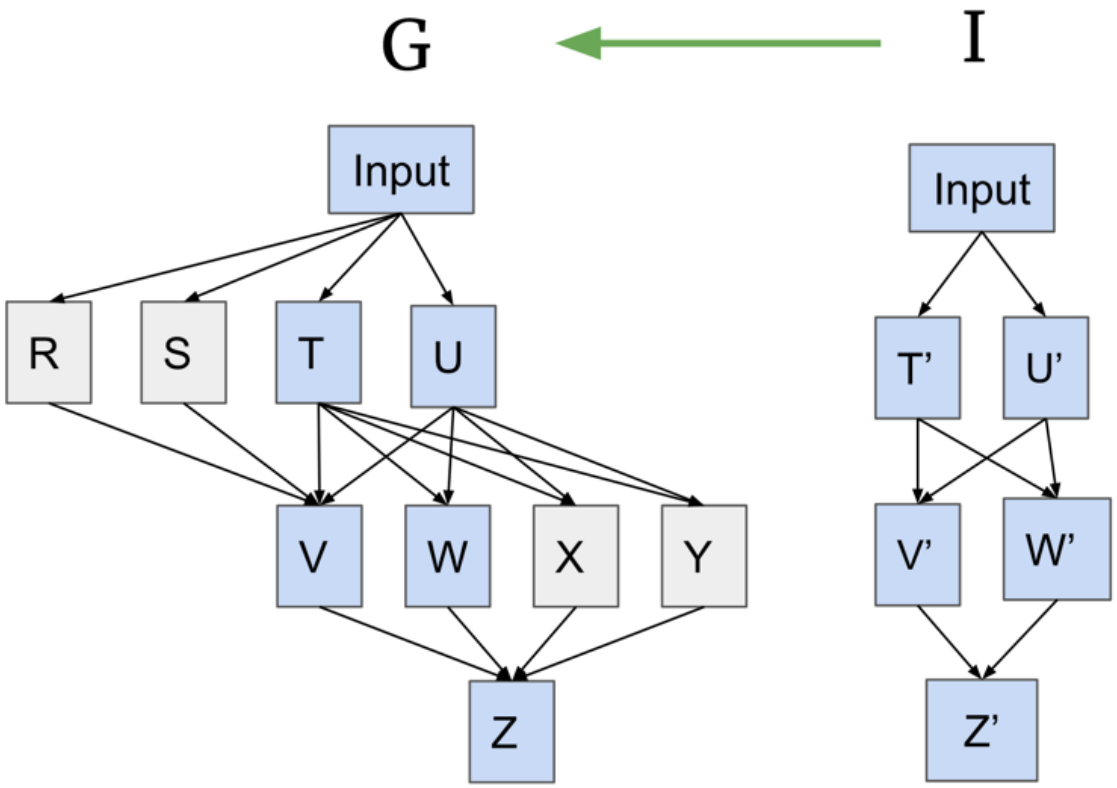
This example, while somewhat unrealistic, hopefully conveys our concern here: sometimes the variation in the outputs of a component is important to your model and performing mean or zero ablation forces this component to only act as a fixed bias term, which is unlikely to be representative of its true contribution to the model's outputs.

We think these examples provide sufficient reasons to be skeptical about the validity of zero or mean ablation and demonstrate our rationale for preferring resampling ablation.

4 Unimportant inputs and isomorphic hypotheses

4.1 Should unimportant inputs be taken from the same or different datapoints?

Suppose we have the following hypothesis where I maps to the nodes of G in blue:



There are four activations in G that we claim are unimportant.

Causal scrubbing requires performing a resampling ablation on these activations. When doing so, should we pick one data point to get all four activations on? Two different data points, one for R and S (which both feed into V) and a different one for X and Y? Or four different data points?

In our opinion, all are reasonable experiments that correspond to subtly different hypotheses. This may not be something you considered when proposing your informal hypothesis, but following the causal scrubbing algorithm forces you to resolve this ambiguity. In particular, the more we sample unimportant activations independently, the more specific the hypothesis becomes, because it allows you to make strictly more swaps. It also sometimes makes it easier for the experimenter to reason about the correlations between different inputs. For a concrete example where this matters, see the paren balance checker experiment.

And so, in the pseudocode above we sample the pairs (R, S) and (X, Y) separately, although we allow hypotheses that require all unimportant inputs throughout the model to be sampled together.[4]

Why not go more extreme, and sample every single unimportant node separately? One reason is that it is not well-defined: we can always rewrite our model to an equivalent one consisting of a different set of nodes, and this would lead to completely different sampling! Another is that we don't actually intend this: we do believe it's important that the inputs to our treeified model be "somewhat reasonable", i.e. have some of the correlations that they usually do in the training distribution, though we're not sure exactly which ones matter. So if

we started from saying that all nodes are sampled separately, we'd immediately want to hypothesize something about them needing to be sampled together in order for our scrubbed model to not get very high loss. Thus this default makes it simpler to specify hypotheses.

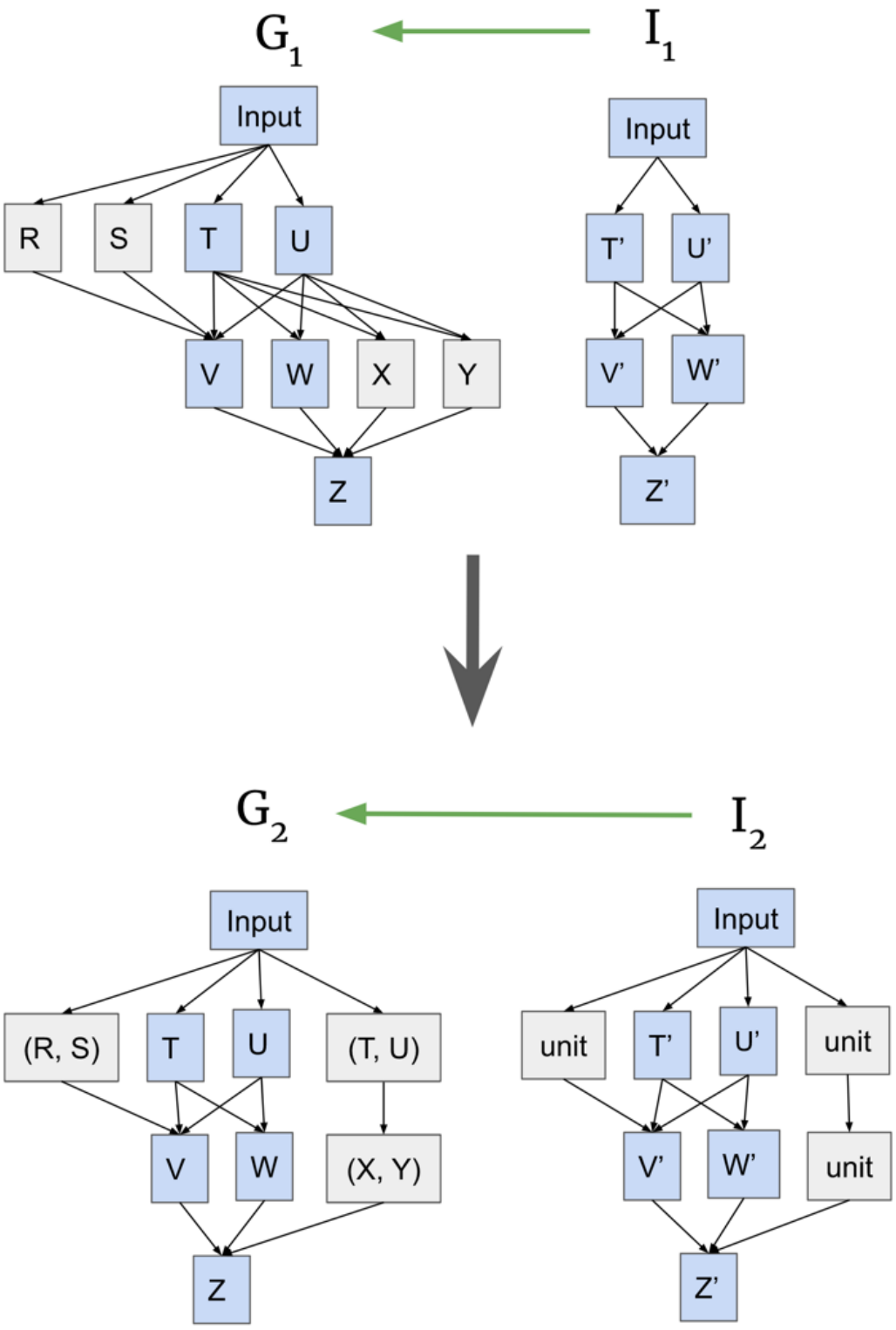
4.2 Including unimportant inputs in the hypothesis

In general we don't require hypotheses to be surjective, meaning not all nodes of G need to be mapped onto by c, nor do we require that G contains all edges of I. This is convenient for expressing claims that some nodes (or edges) of G are unimportant for the behavior. It leaves a degree of freedom, however, in how to treat these unimportant nodes, as discussed in the preceding section.

It is possible to remove this ambiguity by requiring that the correspondence be an isomorphism between G and I. In this section we'll demonstrate how to do this in a way that is consistent with the pseudocode presented, by combining all the unimportant parents of each important node.

In the example below, both R and S are unimportant inputs to the node V, and both X and Y are unimportant inputs to the node Z. We make the following rewrites in the example below:

- If a single important node has multiple unimportant inputs, we combine them. This forms the new node (X, Y) in G_2. We also combine all upstream nodes, such that there is a single path from the input to this new combined node, forming (T, U) which (X, Y) depends on. This ensures we'll only sample one input for all of them in the treeified model.
- We do the same for (R, S) into node V.
- Then we extend I with new nodes to match the entirety of rewritten G. For all of these new nodes that correspond to unimportant nodes (or nodes upstream of unimportant nodes), our interpretation says that all inputs map to a single value (the unit type). This ensures that we can sample any input.
- While we also draw the edges to match the structure of the rewritten G, we will not have other nodes in I be sensitive to the values of these unit nodes.



If you want to take a different approach to sampling the unimportant inputs, you can rewrite the graphs in a different way (for instance, keeping X and Y as separate nodes).

One general lesson from this is that rewriting the computational graphs G and I is extremely expressive. In practice, we have found that with some care it allows us to run the experiments we intuitively wanted to.

5 An alternative formalism: constructing a distribution on treeified inputs

Suppose we have a function f to which we want to apply the causal scrubbing algorithm.

Consider an isomorphic (see above) treeified hypothesis $h = (G_T, I_T, c_T)$ for f. In this appendix we will show that causal scrubbing preserves the joint distribution of inputs to each node of I_T (Lemma 1). Then we show that the distribution of *inputs* induced by causal scrubbing is the maximum entropy distribution satisfying this constraint (Theorem 2).

Let X be the domain of f and D be the input distribution for f (a distribution on X). Let $\widetilde{D}$ be the distribution given by the causal scrubbing algorithm (so the domain of $\widetilde{D}$ is X^n, where n is the number of times that the input is repeated in I_T).

We find it useful to define two sets of random variables: one set for the values of wires (i.e. edges) in I_T when I_T is run on a consistent input drawn from D (i.e. on $(x, \dots, x)$ for some x); and one set for the values of wires in I_T induced by the causal scrubbing algorithm:

Definition (f-consistent random variables): For all the edges of I_T, we call the "f-consistent random variable" the result of evaluating the interpretation I_T on $(x, \dots, x)$, for a random input $x \sim D$. For each node $u \in I_T$, we will speak of the joint distribution of its input wires, and call the resulting random variable the "f-consistent inputs (to u)". We also refer to the value of the wire going out of $u \in I_T$ as the "f-consistent output (to u)".

Definition (scrubbed random variables): Suppose that we run I_T on $(x_1, x_2, \dots, x_n) \sim \widetilde{D}$. In the same way, this defines a set of random variables, which we call the *scrubbed* random variables (and use the terms "scrubbed inputs" and "scrubbed output" accordingly).

Lemma 1: For every node $u \in I_T$, the joint distribution of scrubbed inputs to u is equal to the product distribution of f-consistent inputs to u.

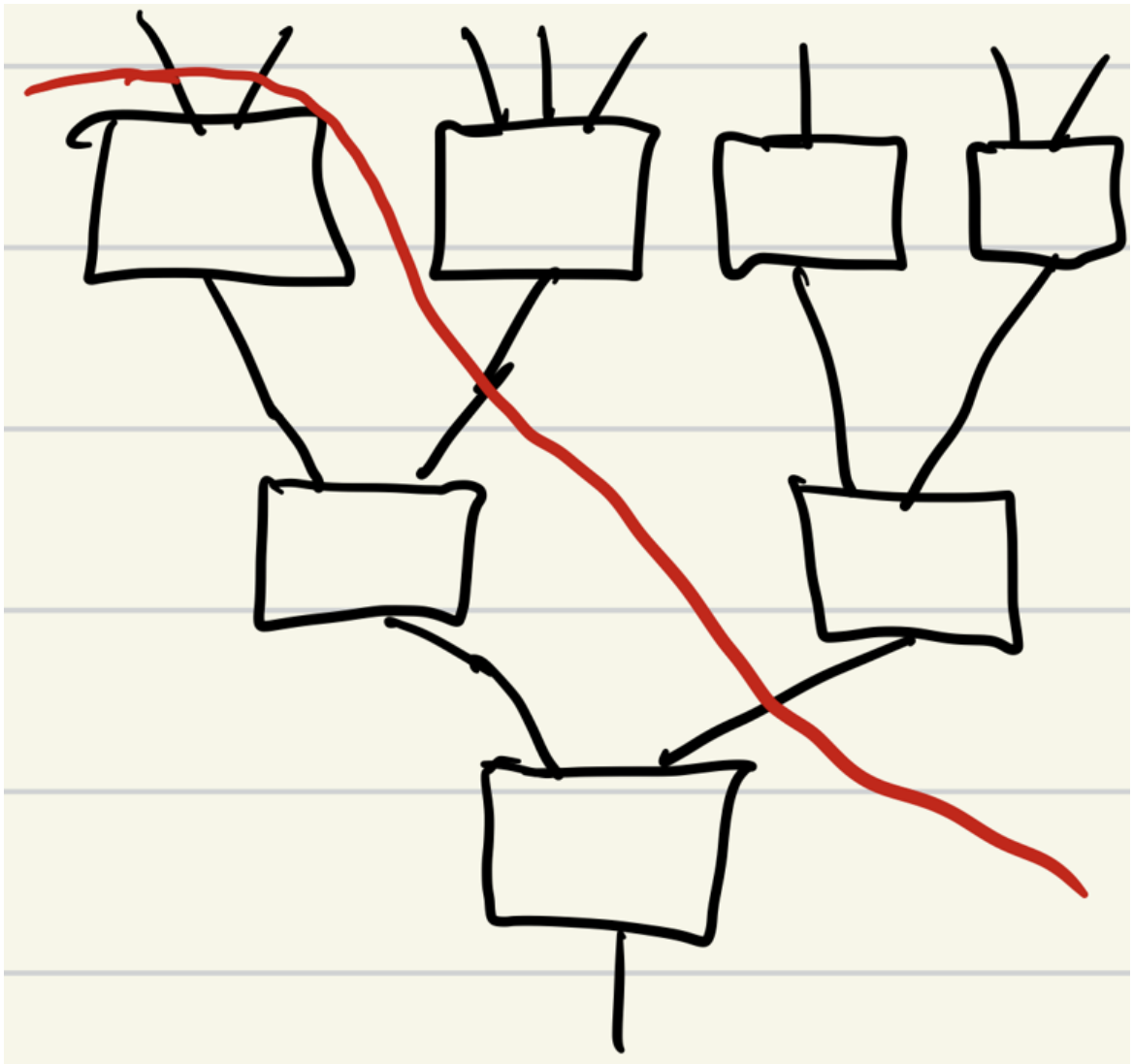
Proof: Recall that the causal scrubbing algorithm assigns a datum in X to every node of I_T, starting from the root and moving up. The key observation is that for every node u of I_T, the distribution of the datum of u is exactly D. We can see this by induction. Clearly this is true

for the root. Now, consider an arbitrary non-root node u and assume that this claim is true for the parent v of u. Consider the equivalence classes on X defined as follows: x_1 and x_2 are equivalent if $(x_1, \dots, x_1)$ has the same value at u as $(x_2, \dots, x_2)$ when I_T is run on each input. Then the datum of u is chosen by sampling from D subject to being in the same equivalence class as the datum of v. Since (by assumption) the datum of v is distributed according to D, so is the datum of u.

Now, by the definition of the causal scrubbing algorithm, for every node u, the scrubbed inputs to u are equal to the inputs to u when I_T is run on the datum of u. Since the datum of u is distributed according to D, it follows that the joint distribution of scrubbed inputs to u is equal to the joint distribution of f-consistent inputs to u.

Theorem 2: The joint distribution of (top-level) scrubbed inputs is the maximum-entropy distribution on X^n, subject to the constraints imposed by Lemma 1.

Proof: We proceed by induction on a stronger statement: consider any way to "cut" through I_T in a way that separates all of the inputs to I_T from the root (and does so minimally, i.e. if any edge is un-cut then there is a path from some leaf to the root). (See below for an example.) Then the joint scrubbed distribution of the cut wires has maximal entropy subject to the constraints imposed by Lemma 1 on the joint distribution of scrubbed inputs to all nodes lying on the root's side of the cut.



An example of a cut

Our base case is the cut through the input wires to the root (in which case Theorem 2 is vacuously true). Our inductive step will take any cut and move it up through some node u, so that if previously the cut passed through the output of u, it will now pass through the inputs of u. We will show that if the original cut satisfies our claim, then so will the new one.

Consider any cut and let u be the node through which we will move the cut up. Let x denote the vector of inputs to u, y be the output of u (so y = u(x)), and z denote the values along all cut wires besides y. Note that x and z are independent conditional on y; this follows by conditional independence rules on Bayesian networks (x and z are d-separated by y).

Next, we show that this distribution is the maximum-entropy distribution. The following equality holds for *any* random variables X, Y, Z such that Y is a function of X:

$$H(X, Z) = H(X, Y, Z) = H(X) + H(Y, Z) - I(X; Y, Z)$$

$$= H(X) + H(Y, Z) - (I(X; Z \mid Y) + I(X; Y))$$

$$= H(X) + H(Y, Z) - H(Y) - I(X; Z \mid Y)$$

Where $I(\cdot)$ is mutual information. The first step follows from the fact that Y is a function of X. The second step follows from the identity $I(A; B) = H(A) + H(B) - H(A, B)$. The third step follows from the identity that $I(A; B \mid C) = I(A; B, C) - I(A; C)$. The last step follows from the fact that $I(X; Y) = H(Y) - H(Y \mid X) = H(Y)$, again because Y is a function of X.

Now, consider all possible distributions of (x, z) subject to the constraints imposed by Lemma 1 on the joint distribution of scrubbed inputs to all nodes lying on the root's side of the updated cut. The lemma specifies the distribution of x and (therefore) y. Thus, subject to these constraints, $H(x, z)$ is equal to $H(y, z) - I(x; z \mid y)$ plus $H(x) - H(y)$, which is a constant. By the inductive hypothesis, $H(y, z)$ is as large as possible subject to the lemma's constraints. Mutual information is non-negative, so it follows that if $I(x; z \mid y) = 0$, then $H(x, z)$ is as large as possible subject to the aforementioned constraints. Since x and z are independent conditional on y, this is indeed the case.

This concludes the induction. So far we have only proven that the joint distribution of scrubbed inputs is *some* maximum-entropy distribution subject to the lemma's constraints. Is this distribution unique? Assuming that the space of possible inputs is finite (which it is if we're doing things on computers), the answer is yes: entropy is a strictly concave function and the constraints imposed by the lemma on the distribution of scrubbed inputs are convex (linear, in particular). A strictly concave function has a unique maximum on a convex set. This concludes the proof.

Fun Fact 3: The entropy of the joint distribution of scrubbed inputs is equal to the entropy of the output of I_T, plus the sum over all nodes $u \in I_T$ of the information lost by u (i.e. the entropy of the joint input to u minus the entropy of the output). (By Lemma 1, this number does not depend on whether we imagine I_T being fed f-consistent inputs or scrubbed inputs.) By direct consequence of the proof of Theorem 2, we have $H(x, z) - H(y, z) = H(x) - H(y)$ (with x, y, z as in the proof of Theorem 2). Proceeding by the same induction as in Theorem 2 yields this fact.

6 How causal scrubbing handles polysemanticity

In [our polysemanticity toy model paper], we introduced an analytically tractable setting where the optimal model represents features in superposition. In this section, we'll analyze this model using causal scrubbing, as an example of what it looks like to handle polysemantic activations.

The simplest form of this model is the two-variable, one-neuron case, where we have independent variables x1 and x2 which both have zero expectation and unit variance, and we are choosing the parameters c and d to minimize loss in the following setting:

$$y = a x_1^2 + b x_2^2$$

$$\tilde{y} = (c x_1 + d x_2)^2 + e$$

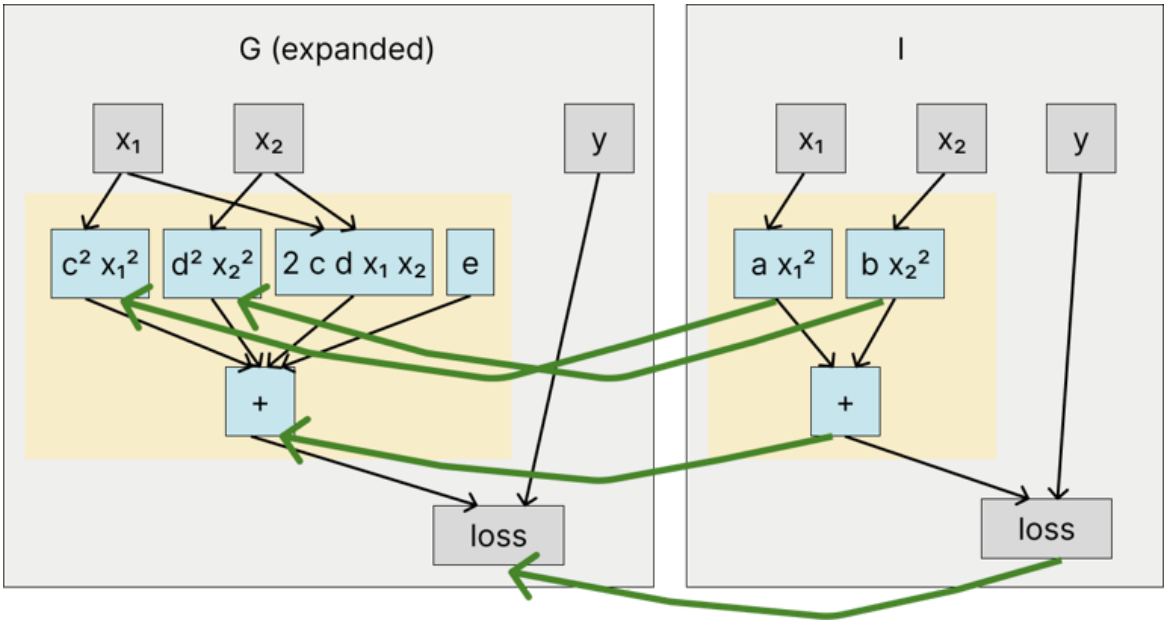
$$\text{loss} = (y - \tilde{y})^2$$

Where $\tilde{y}$ is our model, c and d are the parameters we're optimizing, and a and b are part of the task definition. As discussed in our toy model paper, in some cases (when you have some combination of a and b having similar values and x_1 and x_2 having high kurtosis (e.g. because they are usually equal to zero)), c and d will both be set to nonzero values, and so $(cx_1 + dx_2)$ can be thought of as a superposed representation of both x_1 and x_2.

To explain the performance of this model with causal scrubbing, we take advantage of function extensionality and expand y_tilde:

$$\tilde{y} = c^2 x_1^2 + d^2 x_2^2 + 2 c d x_1 x_2 + e$$

And then we explain it with the following hypothesis:



When we sample outputs using our algorithm here, we're going to sample the interference term from random other examples. And so the scrubbed model will have roughly the same estimated loss as the original model–the errors due to interference will no longer appear on the examples that actually suffer from interference, but the average effect of interference will be approximately reproduced.

In general, this is our strategy for explaining polysemantic models: we do an algebraic rewrite on the model so that the model now has monosemantic components and an error term, and then we say that the monosemantic components explain why the model is able to do the computation that it does, and we say that we don't have any explanation for the error term.

This works as long as the error is actually unstructured–if the model was actively compensating for the interference errors (as in, doing something in a way that correlates with the interference errors to reduce their cost), we'd need to describe that in the explanation in order to capture the true loss.

This strategy also works if you have more neurons and more variables–we'll again write our model as a sum of many monosemantic components and a residual. And it's also what we'd do with real models–we take our MLP or other nonlinear components and make many copies of the set of neurons that are required for computing a particular feature.

This strategy means that we generally have to consider an explanation that's as large as the model would be if we expanded it to be monosemantic. But it's hard to see how we could have possibly avoided this.

Note that this isn't a solution to *finding* a monosemantic basis - we're just claiming that if you had a hypothesized monosemantic reformulation of the model you could test it with causal scrubbing.

This might feel vacuous–what did we achieve by rewriting our model as if it was monosemantic and then adding an error term? We claim that this is actually what we wanted. The hypothesis explained the loss because the model actually was representing the two input variables in a superposed fashion and resigning itself to the random error due to interference. The success of this hypothesis reassures us that the model isn't doing anything more complicated than that. For example, if the model was taking advantage of some relationship between these features that we don't understand, then this hypothesis would not replicate the loss of the model.

6.1 Underestimating interference by neglecting correlations in model errors

Now, suppose we rewrite the model from the form we used above:

$$\tilde{y} = c^2 x_1^2 + d^2 x_2^2 + 2\,c\,d\,x_1\,x_2 + e$$

To the following form:

$$\tilde{y} = c^2 x_1^2 + d^2 x_2^2 + c\,d\,x_1\,x_2 + c\,d\,x_1\,x_2 + e$$

Where we've split the noise term into two pieces. If we sample these two parts of the noise term independently, we will have effectively reduced the magnitude of the noise, for the usual reason that averages of two samples from a random variable have lower variance than

single samples. And so if we ignore this correlation, we'll estimate the cost of the noise to be lower than it is for the real model. This is another mechanism by which ignoring a correlation can cause the model to seem to perform better than the real model does; as before, this error gives us the opportunity to neglect some positive contribution to performance elsewhere in the model.

7 An additional example of approving a false hypothesis

We can construct cases where the explanation can make the model look better by sneaking in information. For example, consider the following setting:

The model's input is a tuple of a natural number and the current game setting, which is either EASY or HARD (with equal frequency). The model outputs the answer either "0", "1", or "I don't know". The task is to guess the last bit of the hash of the number.

Here's the reward function for this task:

Game mode	Score if model is correct	Score if model is incorrect	Score if model says "I don't know"
EASY	2	-1	0
HARD	10	-20	0

If the model has no idea how to hash numbers, its optimal strategy is to guess when in EASY mode and say "I don't know" in HARD mode.

Now, suppose we propose the hypothesis that claims that the model outputs:

- on an EASY mode input, what the model would guess; and
- on a HARD mode input, the correct answer.

To apply causal scrubbing, we consider the computational graph of both the model and the hypothesis to consist of the input nodes and a single output node. In this limited setting, the projected model runs the following algorithm:

- Replace the input with a random input that would give the same answer according to the hypothesis; and
- Output what the model outputs on that random input.

Now consider running the projected model on a HARD case. According to the hypothesis, we output the correct answer, so we replace the input

- half the time with another HARD mode input (with the same answer), on which the model outputs "I don't know"; and
- half the time with an EASY mode input chosen such the model will guess the correct answer.

So, when you do causal scrubbing on HARD cases, the projected model will now guess correctly half the time, because half its "I don't know" answers will be transformed into the correct answer. The projected model's performance will be worse on the EASY cases, but the HARD cases mattered much more, so the projected model's performance will be much better than the original model's performance, even though the explanation is wrong!

In examples like this one, hypotheses can cheat and get great scores while being very false.

8 Adversarial validation might be able to elicit true hypotheses

(Credit for the ideas in this section is largely due to ARC.)

We might have hoped that we'd be able to use causal scrubbing as a check on our hypotheses analogous to using a proof checker like Lean or Coq to check our mathematical proofs, but this doesn't work. Our guess is that it's probably impossible to have an efficient algorithm for checking interpretability explanations which always rejects false explanations. This is mostly because we suspect that interpretability explanations should be regarded as an example of defeasible reasoning. Checking interpretations in a way that rejects all false explanations is probably NP-hard, and so we want to choose a notion of checking which is weaker.

We aren't going to be able to check hypotheses by treating as uncorrelated everything that the hypotheses claimed wasn't relevantly correlated. This would have worked if ignoring correlations could only harm the model. But as shown above, we have several cases where ignoring correlations helps the model.

So we can't produce true explanations by finding hypotheses subject to the constraint that they predict the observed metrics. As an alternative proposal, we can check if hypotheses are comprehensive by seeing if any adversarial additions to the hypothesis would cause the predicted metric to change considerably. In all of the counterexamples above, the problem is that the metric was being overestimated because there were important correlations that were being neglected and which would reduce the estimated metric if they were included. If we explicitly check for additional details to add to our hypotheses which cause the estimated metric to change, all the counterexamples listed above are solved.

To set up this adversarial validation scheme, we need some mechanism for hypotheses to be constructed adversarially. That is, we need to handle cases where the adversary wants to rewrite f to an extensionally-equal function. One way of thinking about this is that we want a function `join` which is a binary operation on hypotheses, taking the two hypotheses to the hypothesis which preserves all structure in the model that either of the two hypotheses preserved.

Here are two ways of defining this operation:

- **Swap-centric.** You can think of a hypothesis as a predicate on activation swaps (of the same activation on two different inputs). From this perspective, you can define join(h1, h2) to be the hypothesis which permits a swap iff h1 and h2 both permit it.
- **Computation graph centric.** You can equivalently construct the joined hypothesis by the following process. First, ensure that each of the correspondences are bijections, and that both I_1 and I_2 have the same shape, adding extra no-op nodes as necessary.

  Now we can define I of the joined hypothesis to be the graph where every node contains the tuple of the values from the two earlier interpretations.

The main failure of the algorithm listed above is that we don't know how to handle cases where the adversary wants to rewrite f to an extensionally-equal function in a way which is mutually incompatible with the original hypothesis (for example, because their computational graphs have different shapes and there's no way to splice the two computational graphs together). This is a pretty bad problem because the function

extensionality move seems very important in practice. ARC has worked on basically this problem for a while and hasn't yet solved it, regrettably.

Some other questions that we haven't answered:

- How do we incentivize specific explanations? We don't know (but haven't thought about it that much). Our current proposals look something like having a budget for how much hypotheses can reduce entropy.
- The explanations produced by this process will probably by default be impossible for humans to understand; is there some way to fix this? We also don't have good ideas here. (Note that this isn't a failure that's specific to causal scrubbing; it seems fundamentally challenging to generate human-understandable interpretations for complicated superhuman models.) That being said, a lot of our optimism about interpretability comes from applications where the interpretability tools are used by AIs or by human-coded algorithms, rather than by humans, so plausibly we're fine even if humans can't understand the interpretability results.

Overall, it seems plausible that these problems can be overcome, but they are definitely not currently solved. We hold out hope for an interpretability process which has validity properties which allow us to use powerful optimization inside it and still trust the conclusions, and hope to see future work in this direction.

1. ^

   This is also true when you're training models with an autodiff library–you construct a computational graph that computes loss, and run backprop on the whole thing, which quickly recurses into the model but doesn't inherently treat it differently.

2. ^

   This allows for testing out human interpretable approximations to neural network components: 'Artificial Artificial Neural networks'. We think it's more informative to see how the model performs with the residual of this approximation resampling ablated as opposed to zero ablated.

3. ^

   In general, you could have the output be non-scalar with any distance metric δ to evaluate the deviation of the scrubbed expectation, but we'll keep things simple here.

4. ^

   Another way of thinking about this is: when we consider the adversarial game setting, we would like each side to be able to request that terms are sampled together. By default therefore we would like terms (even random ones!) to be sampled separately.

Causal scrubbing: results on induction heads

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

* *Authors sorted alphabetically.*

This is a more detailed look at our work applying causal scrubbing to induction heads. The results are also summarized here .

Introduction

In this post, we'll apply the causal scrubbing methodology to investigate how induction heads work in a particular 2-layer attention-only language model.[1] While we won't end up reaching hypotheses that are fully specific or fully human-understandable, causal scrubbing will allow us to validate claims about which components and computations of the model are important.

We'll first identify the induction heads in our model and the distribution over which we want to explain these heads' behavior. We'll then show that an initial naive hypothesis about how these induction heads mechanistically work—similar to the one described in the Induction Heads paper (Olsson et al 2022)—only explains 35% of the loss. We'll then go on to verify that a slightly more general, but still reasonably specific, hypothesis can explain 89% of the loss. It turns out to be the case that induction heads in this small model use information that is flowing through a variety of paths through the model – not just previous token heads and the embeddings. However, the important paths can be constrained considerably – for instance, only a small number of sequence positions are relevant and the way that attention varies in layer 0 is not that important.

As with the paren balance checker post, this post is mostly intended to be pedagogical. You can treat it as an initial foray into developing and testing a hypothesis about how induction heads work in small models. We won't describe in detail the exploratory work we did to produce various hypotheses in this document; we mostly used standard techniques (such as looking at attention patterns) and causal scrubbing itself (including looking at internal activations from the scrubbed model such as log-probabilities and attention patterns).

Methodology

Overall approach

The experiments prescribed by causal scrubbing in this post are roughly equivalent to performing resampling ablations on the parts of the rewritten model that we claim are unimportant. For each 'actual' datum we evaluate the loss on, we'll always use a single 'other' datum for this resampling ablation.

Throughout this document we measure hypothesis quality using the percentage of the loss that is recovered under a particular hypothesis. This percentage may exceed 100% or be negative, it's not actually a fraction. See the relevant section in the appendix post for a formal definition.

Note that, in these examples, we're not writing out formal hypotheses, as defined in our earlier post, because the hypotheses are fairly trivial while also being cumbersome to work with. In brief, our I is identical to G with all the edges we say don't matter removed, and every node computing the identity.

Model architecture

We studied a 2-layer attention-only model with 8 heads per layer. We use L.H as a notation for attention heads where L is the zero-indexed layer number and H is the zero-indexed head number.

Further details about the model architecture (which aren't relevant for the experiments we do) can be found in the appendix.

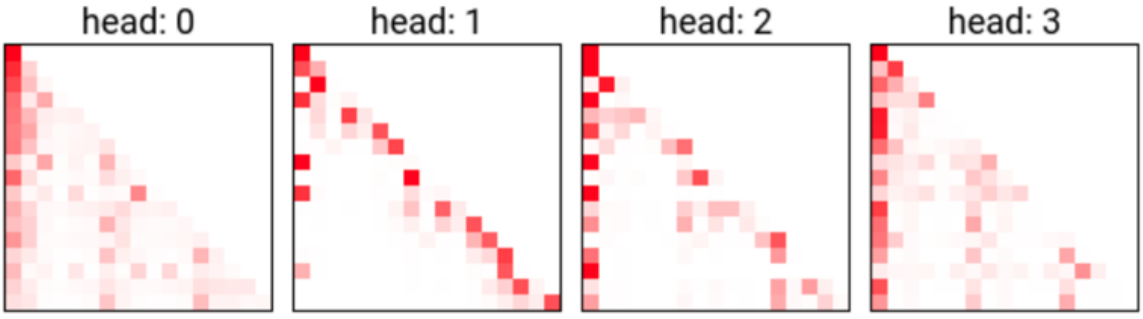
Identification

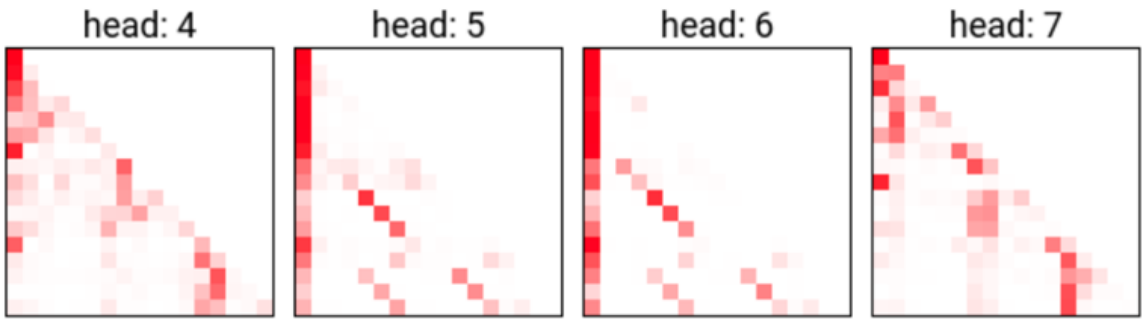
Identifying induction heads

Induction heads, originally described in A Mathematical Framework for Transformer Circuits (Elhage et al 2021), are attention heads which empirically attend from some token `[A]` back to earlier tokens `[B]` which follow a previous occurrence of `[A]`. Overall, this looks like `[A][B]...[A]` where the head attends back to `[B]` from the second `[A]`.

Our first step was to identify induction heads. We did this by looking at the attention patterns of layer 1 heads on some text where there are opportunities for induction. These heads often either attend to the first token in a sequence, if the current token doesn't appear earlier in the context, or look at the token following the previous occurrence of the current token.

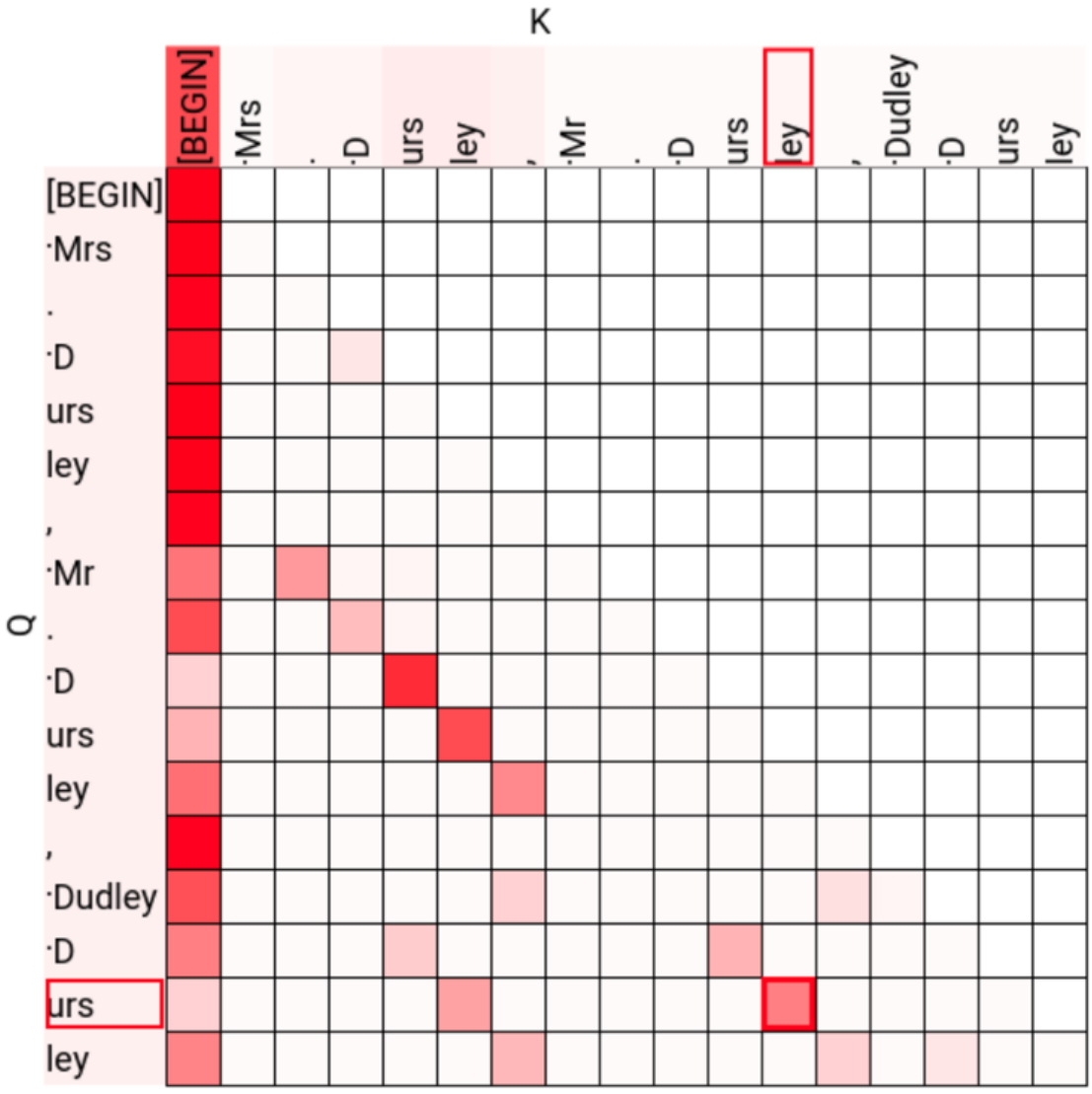
Here are all the attention patterns of the layer 1 heads on an example sequence targeted at demonstrating induction: "Mrs. Dursley, Mr. Dursley, Dudley Dursley"





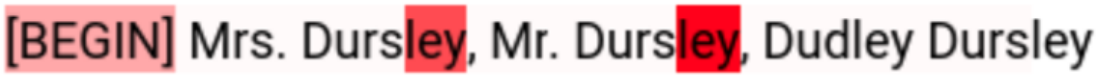
Link to the interactive interface that produced this plot and many others in this notebook. Feel free to explore!

Two heads seem like possible induction heads: 1.5 and 1.6. We can make this more clear by looking more closely at their attention patterns: for instance, zooming in on the attention pattern of 1.6 we find that it attends to the sequence position corresponding to the last occurrence of `"[ley]"`:



Link to interactive interface

Within this, let's specifically look at the attention from the last 'urs' token (highlighted in the figure above).

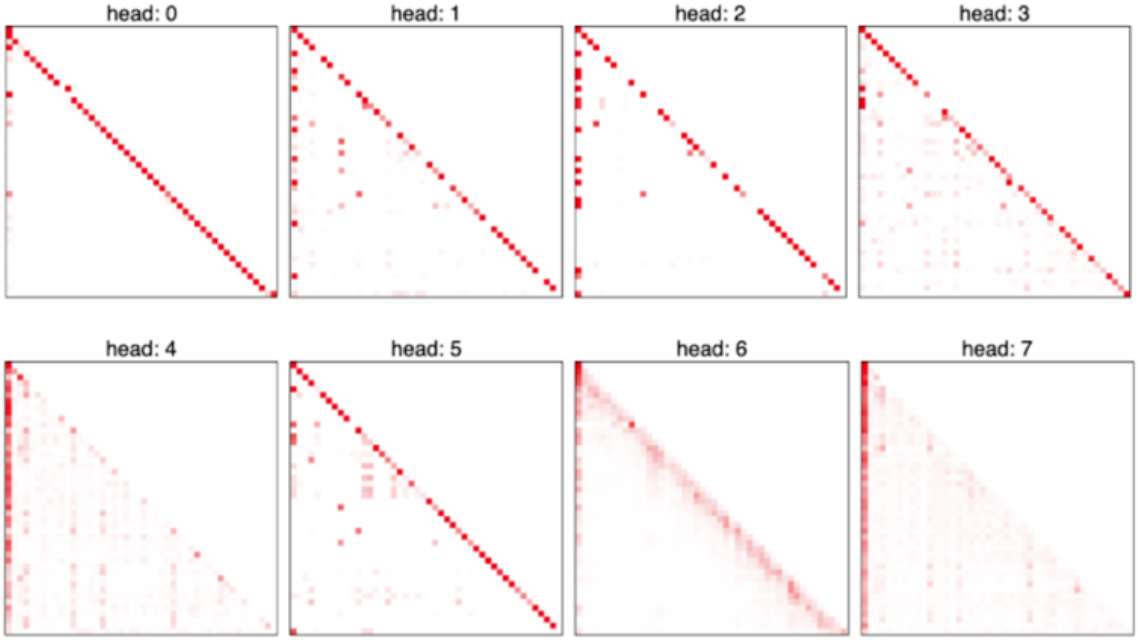


Link to interactive interface

A closer look at the attention pattern of head 1.5 showed similar behavior.

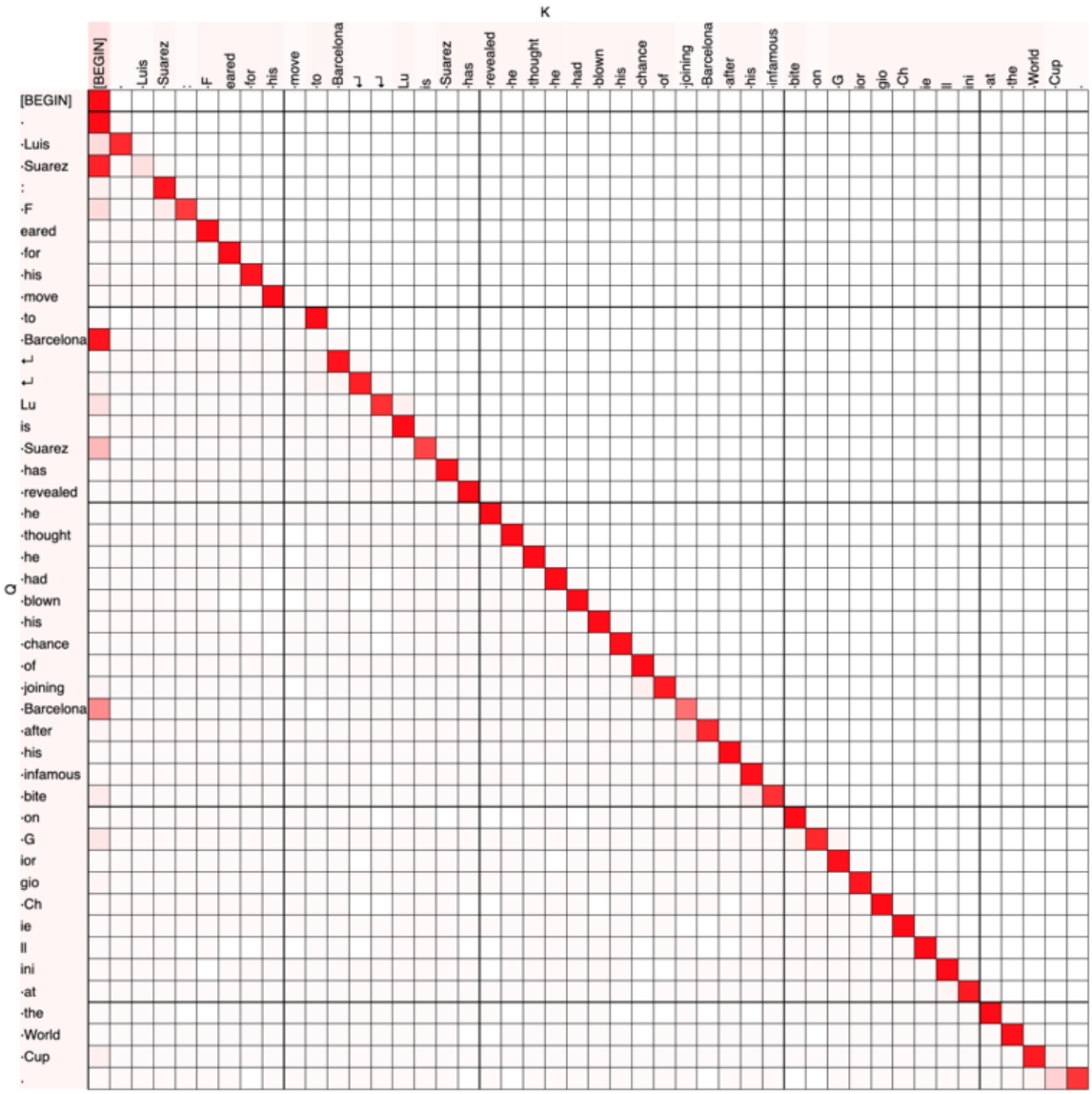
Identifying the previous token head

Previous token heads are heads that consistently attend to the previous token. We picked out the head that we thought was a previous token head by eyeballing the attention patterns for the layer zero heads. Here are the attention patterns for the layer zero heads on a short sequence from OpenWebText:



Link to interactive interface

Here's a plot of 0.0's attention pattern:



[Link to interactive interface]

So you can see that 0.0 mostly attends to the previous token, though sometimes attends to the current token (e.g. on “ to”) and sometimes attends substantially to the `[BEGIN]` token (e.g. from “ Barcelona”).

Picking out tokens at which the model is particularly likely to do induction

Let's define a "next-token prediction example" to be a context (a list of tokens) and a next token; the task is to predict the next token given the context. (Normally, we train autoregressive language models on all the prefixes of a text simultaneously, for performance reasons. But equivalently, we can just think of the model as being trained on many different next-token prediction examples.)

We made a bunch of next-token prediction examples in the usual way (by taking prefixes of tokenized OWT documents), then filtered to the subset of these examples where the last

token in the context was in a particular whitelist of tokens.

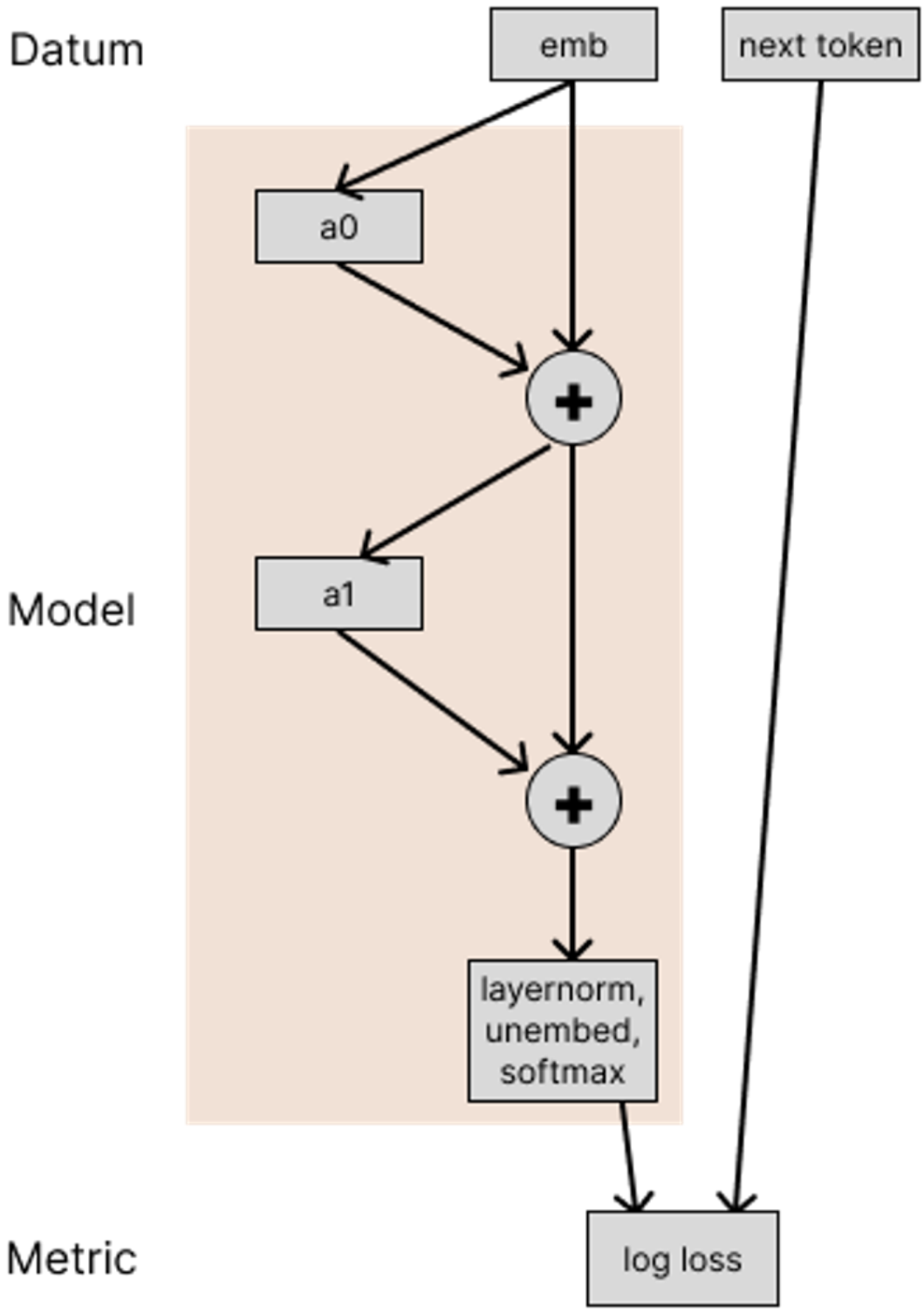
We chose this whitelist by following an approach which is roughly 'select tokens such that hard induction is very helpful over and above bigrams'--see the appendix for further details. Code for this token filtering can be found in the appendix and the exact token list is linked. Our guess is that these results will be fairly robust to different ways of selecting the token whitelist.

So, we didn't filter based on whether induction was a useful heuristic on this particular example, or on anything about the next-token; we only filtered based on whether the last token in the context was in the whitelist.

For all the hypotheses we describe in this post, we'll measure the performance of our scrubbed models on just this subset of next-token prediction examples. The resulting dataset is a set of sequences whose last token is somewhat selected for induction being useful. Note that evaluating hypotheses on only a subset of a dataset, as we do here, is equivalent to constructing hypotheses that make no claims on tokens other than our "inductiony" tokens, and then evaluating these weaker hypotheses on the whole dataset.

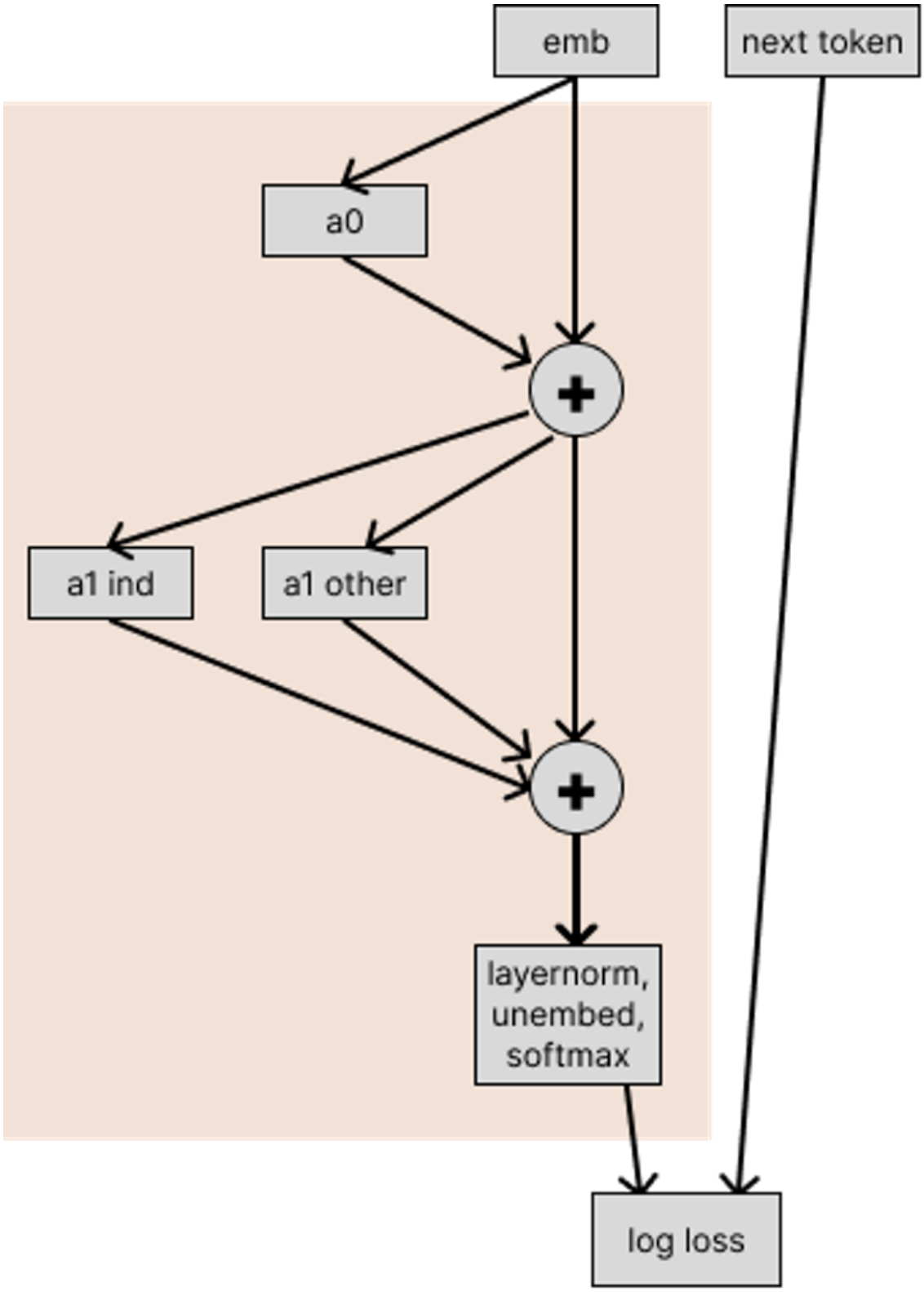
Establishing a baseline

We want to explain the performance of our two-layer attention-only model. Its performance is measured by the following computational graph:

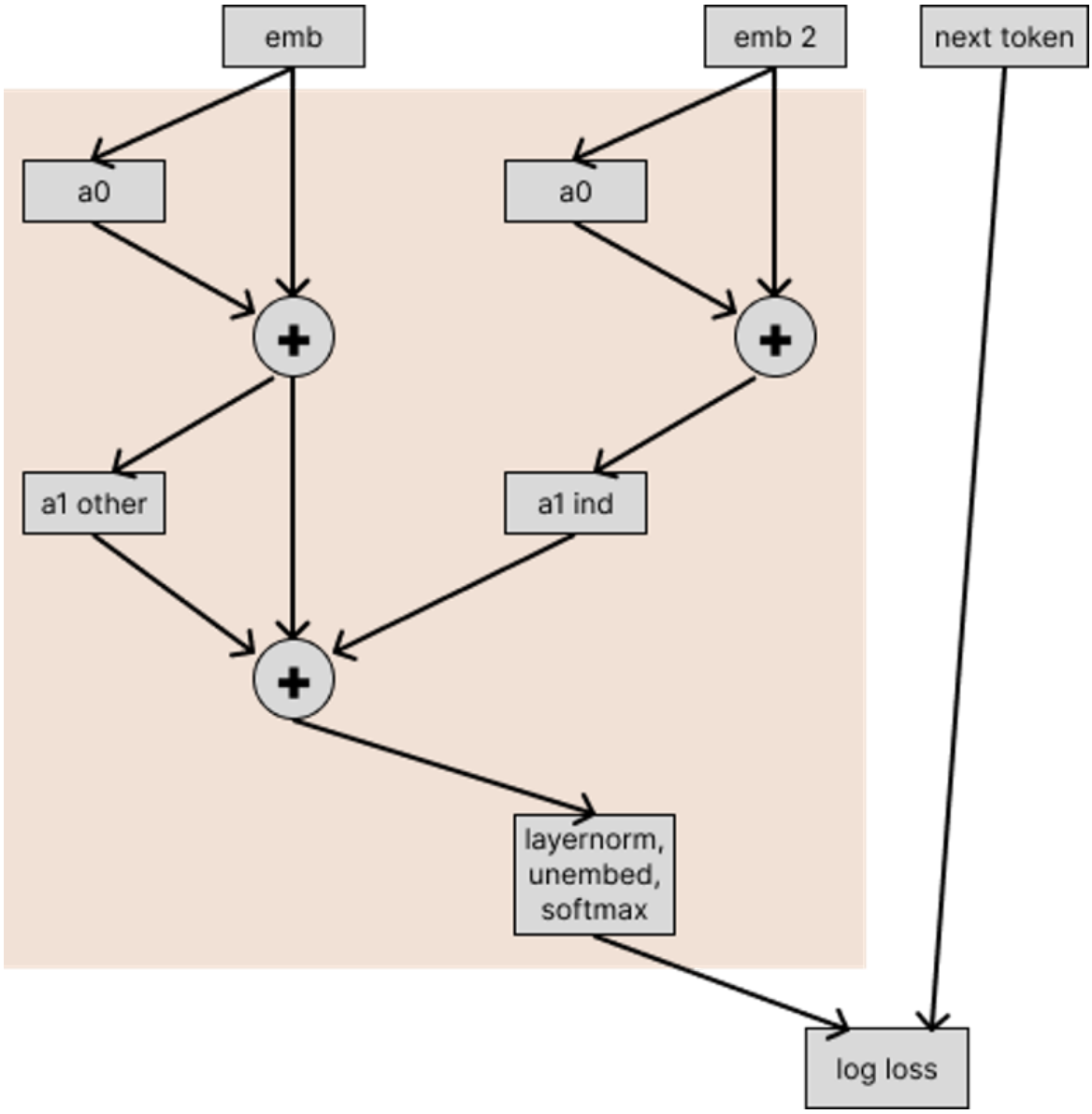


We’re taking the token embeddings (emb) and running them through the model, then calculating the log-loss of the model on the actual next token. The model is composed of two attention layers (with layernorm), which we’re writing as a0 and a1.

To start out our investigation, let’s see how much performance is damaged if we replace the induction head’s outputs with their outputs on random other sequences. To measure this, we rewrite our model to separate out the induction heads from the other layer 1 heads:



Now we consider passing in different inputs into the induction heads and the other heads.



We run the model *without the induction heads* on the (emb, next token) pairs that correspond to sequences in our filtered next-token-prediction dataset, while running the induction heads on a different sequence (encoded as emb2).

Note that if we ran this computational graph with emb2 = emb, we'd exactly recover the performance of the original model–we've rewritten the model to be exactly mathematically equal, except that we're now able to pass in inputs that are different for different paths through the model.

If the induction heads don't contribute at all to model performance on this task, this experiment would return the same loss as the original model.

When you run the scrubbed computation, the loss is 0.213. The original loss on this dataset was 0.160, and the difference between these losses is 0.053. This confirms that the induction

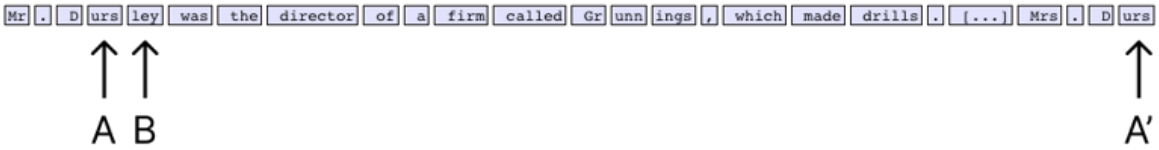
heads contribute significantly to the performance of the original model for this subset of tokens.

Going forward, we'll report the fraction of this 0.053 loss difference that is restored under various scrubs.

For every experiment in this post, we use the same choice of emb2 for each (emb, next token) pair. That is, every dataset example is paired with a single other sequence[2] that we'll patch in as required; in different experiments, the way we patch in the other sequence will be different, but it will be the same other sequence every time. We do this to reduce the variance of comparisons between experiments.

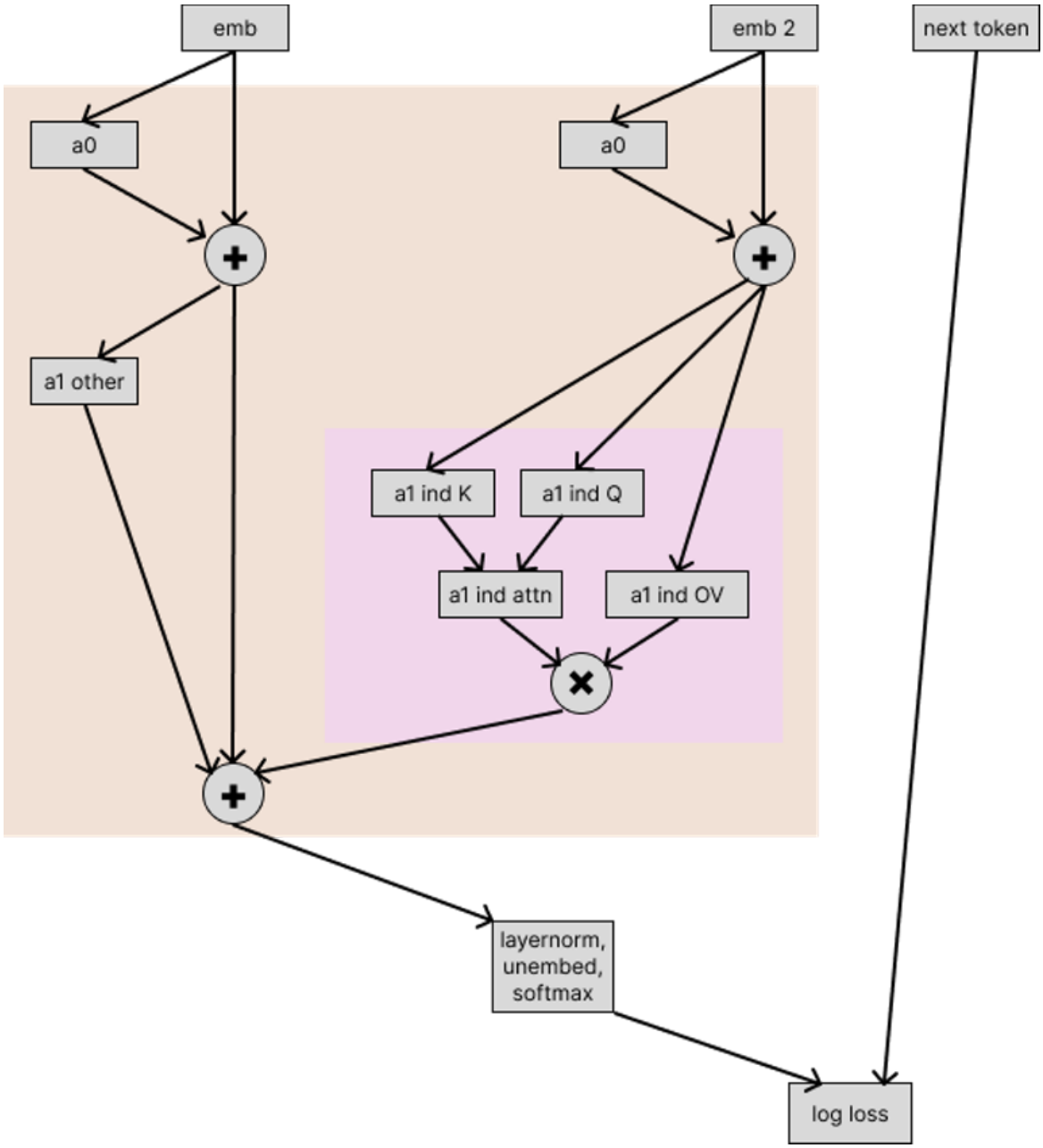
Initial naive hypothesis

This is the standard picture of induction:



- We have a sequence like "Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Durs". "Dursley" is tokenized as | D|urs|ley|. And so a good prediction from the end of this sequence is "ley". (We'll refer to the first "urs" token as A, the first "ley" token as B, and the second "urs" token as A'.)
- There's a previous-token head in layer 0 which copies the value at A onto B.
- The induction head at A' attends to B because of an interaction between the token embedding at A' and the previous-token head output at B.
- The induction head then copies the token embedding of B to its output, and therefore the model proposes B as the next token.

To test this, we need to break our induction heads into multiple pieces that can be given inputs separately. We first expand the node (highlighted in pink here):



So we've now drawn the computation for the keys, queries, and values separately. (We're representing the multiplications by the output matrix and the value matrix as a single "OV" node, for the same reasons as described in the "Attention Heads are Independent and Additive" section of A Mathematical Framework for Transformer Circuits.)

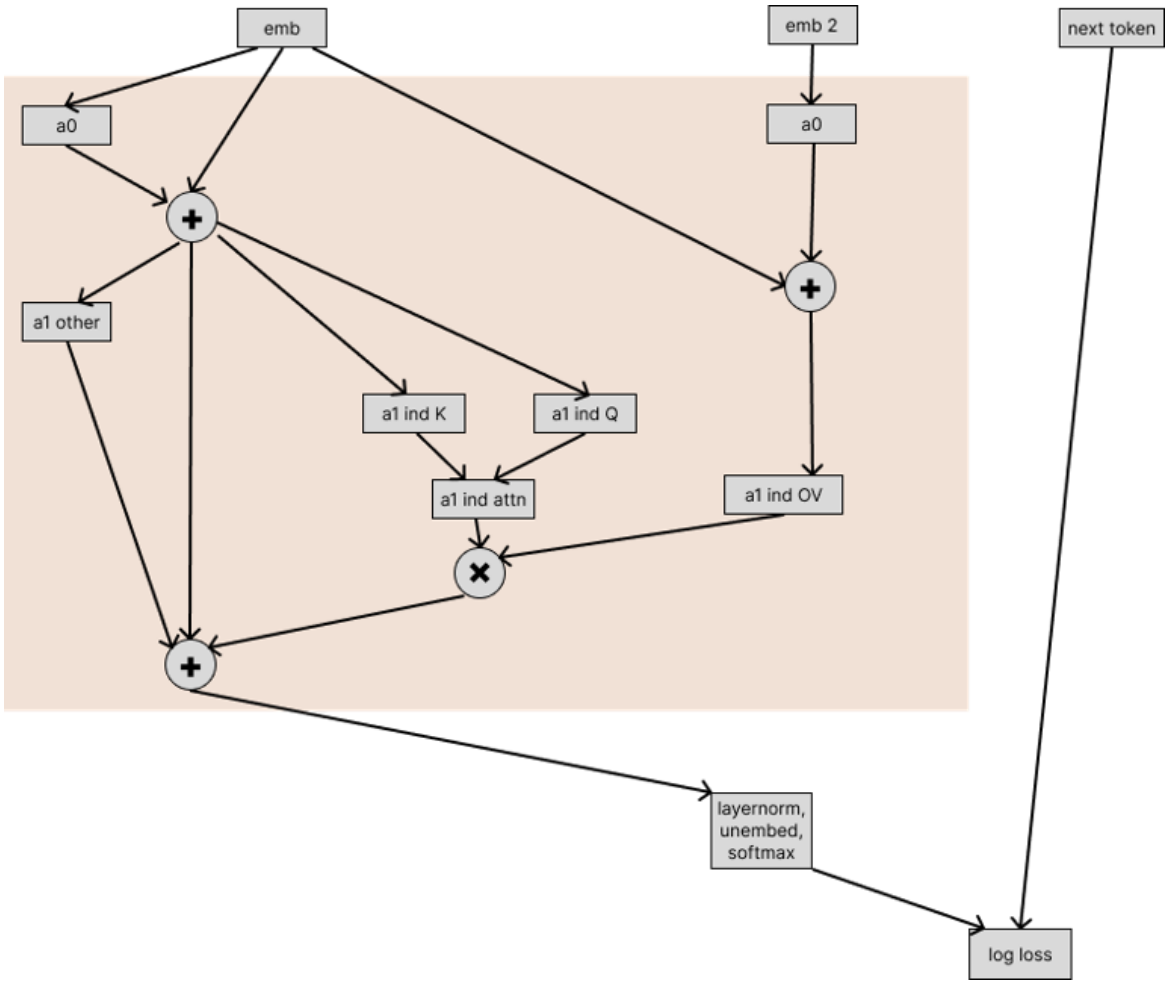
Our hypothesis here involves claims about how the queries, keys, and values are formed:

- values for the induction head are produced only from the token embeddings via the residual stream with no dependence on a0
- queries are also produced only from the token embeddings
- keys are produced only by the previous-token head

Before we test them together, let's test them separately.

The embeddings → value hypothesis

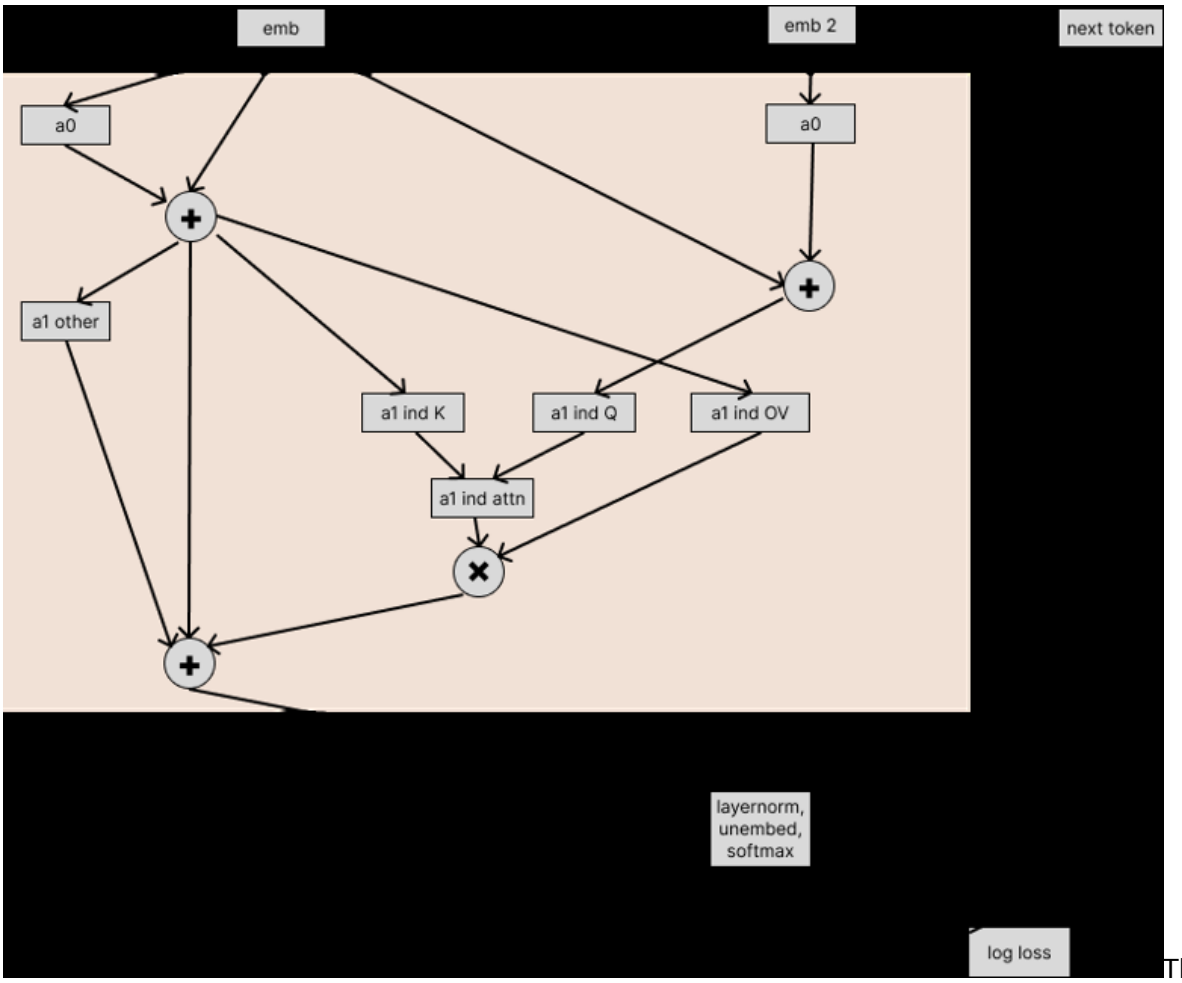
The hypothesis claims that the values for the induction head are produced only from the token embeddings via the residual stream, with no dependence on a0. So, it it shouldn't affect model behavior if we rewrite the computation such that the a1 induction OV path is given the a0 output from emb2, and so it only gets the information in emb via the residual connection around a0:



When we do this scrub, the measured loss is 90% of the way from the baseline ablated model (where we ran the induction heads on emb2) to the original unablated model. So the part of the hypothesis where we said only the token embeddings matter for the value path of the induction heads is somewhat incorrect.

The embeddings → query hypothesis

We can similarly try testing the "the queries for induction heads are produced only from the token embeddings" hypothesis, with the following experiment:

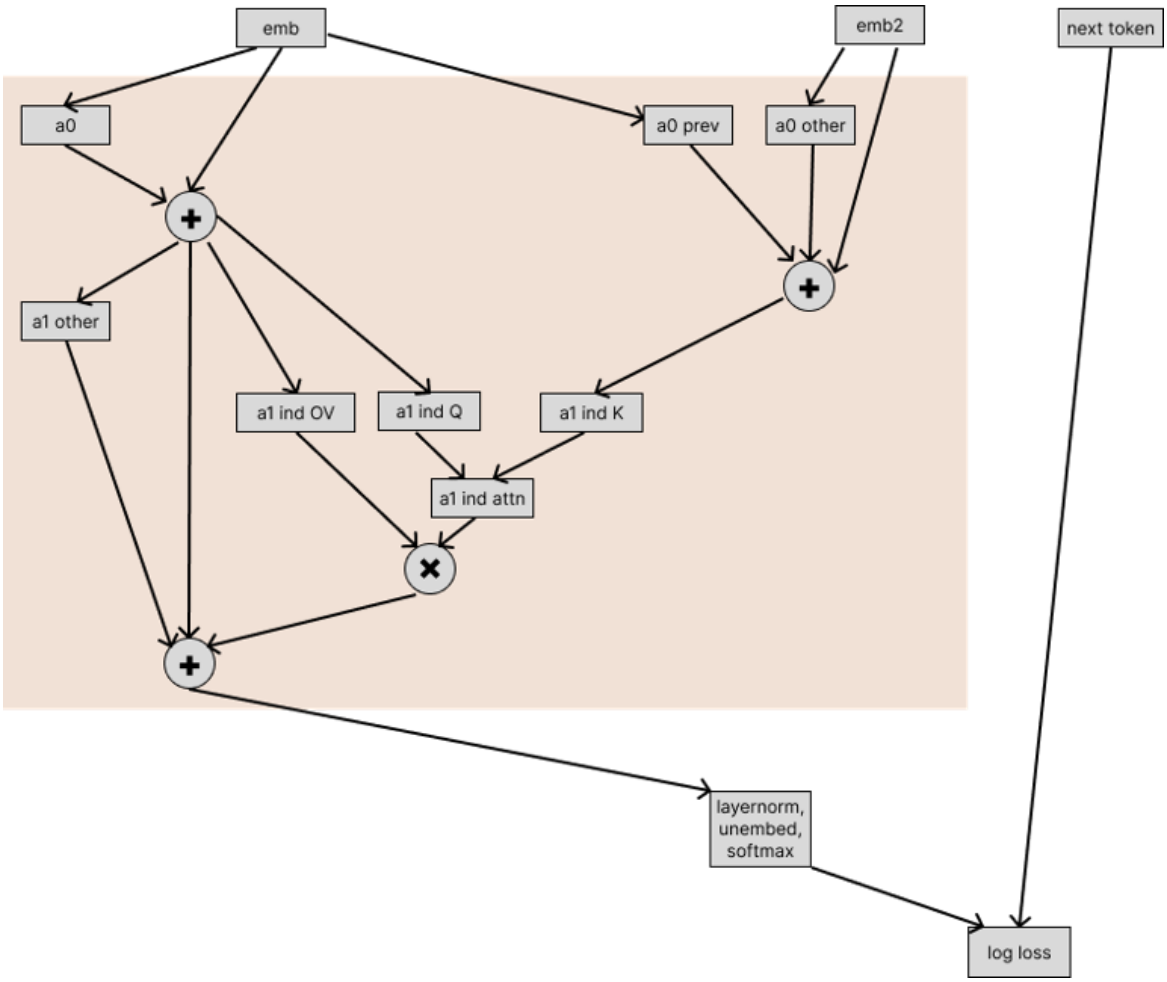


The fraction of the loss restored in this experiment is 51%, which suggests that this part of the hypothesis was substantially less correct than the part about how the induction head values are produced.

The previous-token head → key hypothesis

Finally, we want to test the final claim in our hypothesis; that the key used by the induction head is produced only by the previous-token head.

To do this, we first rewrite our computational graph so that the induction key path takes the previous-token head separately from the other layer zero heads.



This experiment here aims to evaluate the claim that the only input to the induction heads that matters for the keys is the input from the previous-token head.

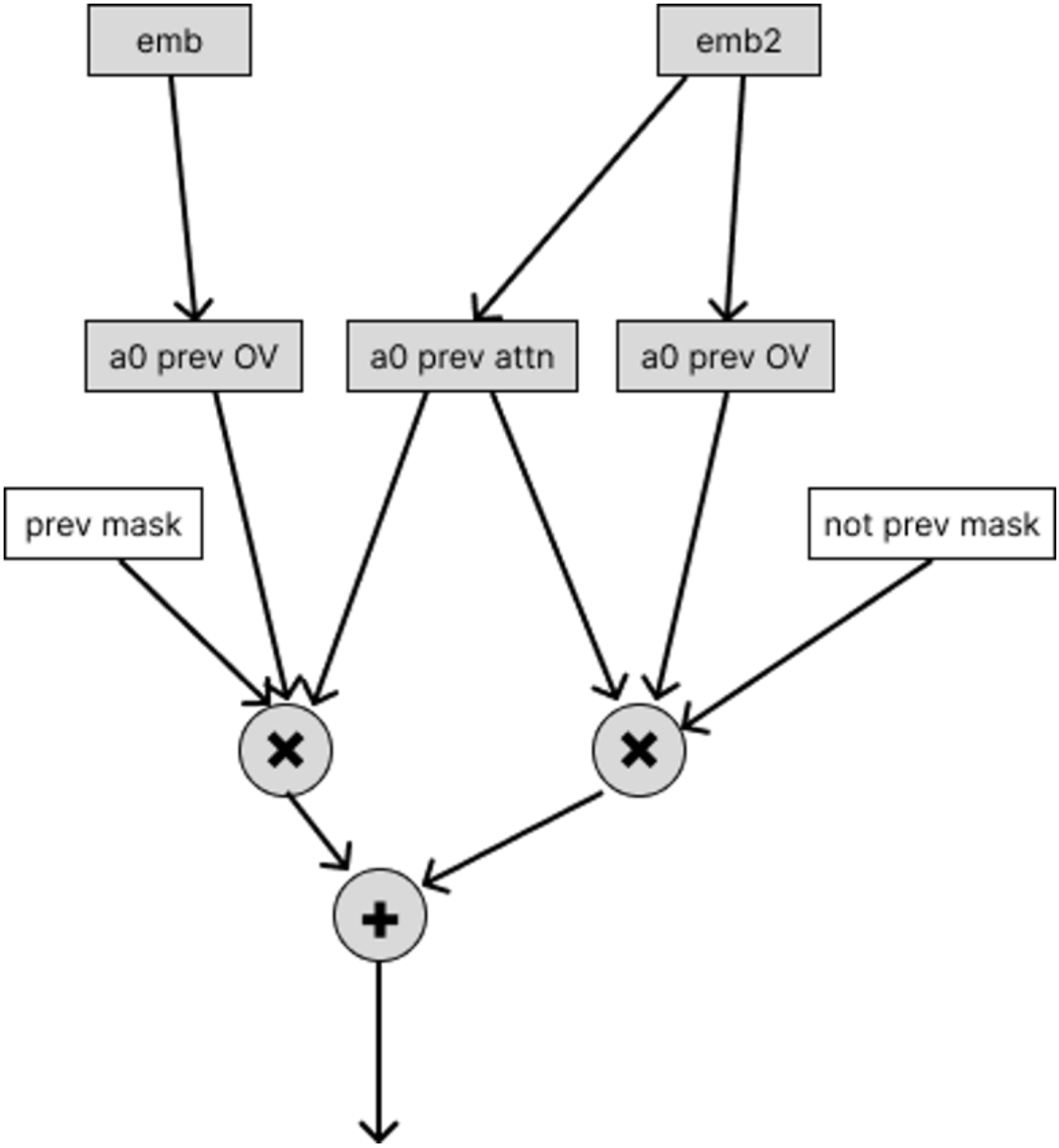
However, this experiment wouldn't test that the previous-token head is actually a *previous* token head. Rather, it just tests that this particular head is the one relied on by the induction heads.

We can make a strong version of this previous token head claim via two sub-claims:

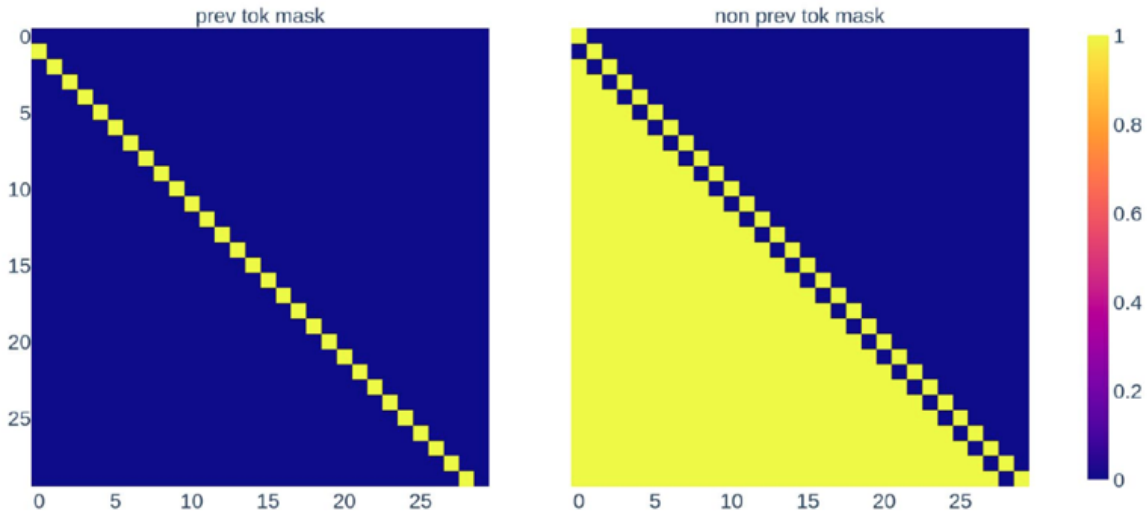
- The attention pattern is unimportant (by which we mean that the relationship between the attention pattern and the OV is unimportant, as discussed in this section of our earlier post)
- All that matters for the OV is the previous sequence position

We'll implement these claims by rewriting the model to separate out the parts which we claim are unimportant and then scrubbing these parts. Specifically, we're claiming that this head always operates on the previous token through its OV (so we connect that to "emb"); and its attention pattern doesn't depend on the current sentence (so we connect that to "emb2"). We also connect the OV for tokens that are not the previous one to "emb2".

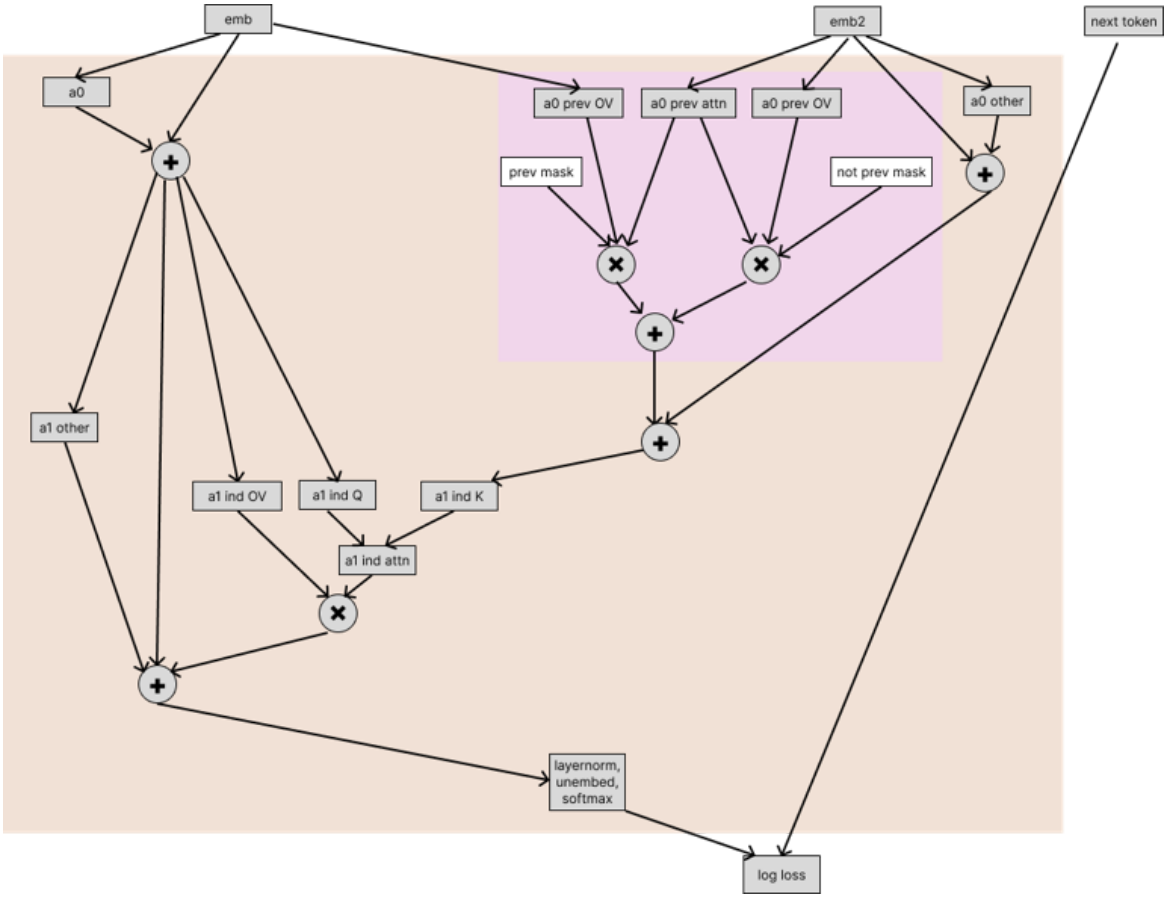
The resulting computation for the previous-token head is as follows:



So we've run the OV circuit on both emb and emb2, and then we multiply each of these by a mask so that we only use the OV result from emb for the previous token. Prev mask is a matrix that is all zeros except for the row below the diagonal (corresponding to attention to the previous token). Non prev mask is the difference between prev mask and the lower triangular mask that we normally use to enforce that attention only looks at previous sequence positions.



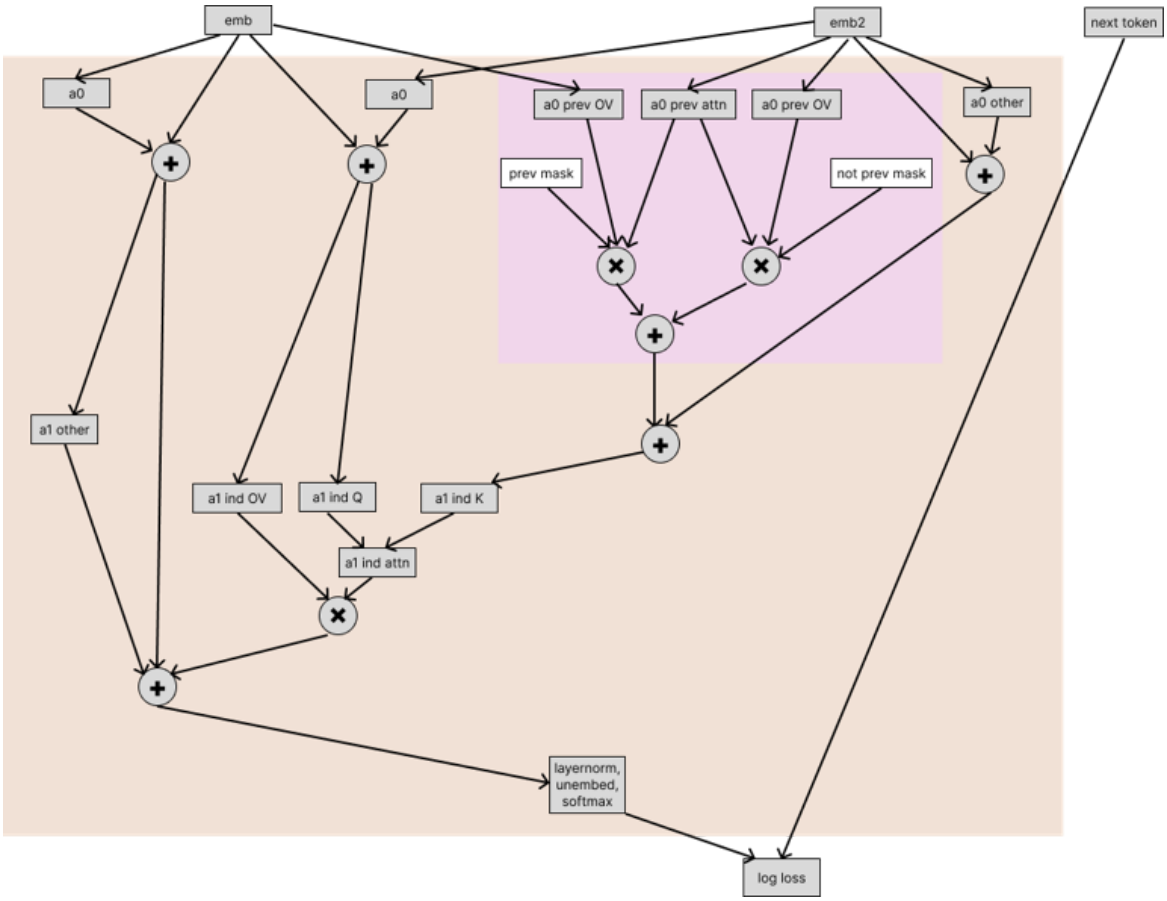
And so, our overall experiment is as follows, where the nodes of the model corresponding to the previous token head are shown in pink:



This fraction of the loss restored by this experiment is 79%.

Scrubbing these all together

Next we want to scrub all these paths (i.e. do all these interventions) simultaneously.



The fraction of the loss that this restores is 35%.

Takeaways

Using causal scrubbing, we've found that our initial naive hypothesis is quite incorrect for these induction heads.

To recap the results, the fractions of loss restored are:

- Scrubbing all of the input to Q except the embeddings: 51%.
- Scrubbing all of the input to K, except the previous token part of the previous-token head: 79%
- Scrubbing all of the input to V except the embeddings: 90%
- Doing all of these at once: 35%

These numbers weren't very surprising to us. When we described this experiment to some of the authors of the induction heads paper, we asked them to guess the proportion of loss that this would recover, and their answers were also roughly in the right ballpark.

Refined Hypotheses

Refined hypothesis 1

How might our previous hypothesis be missing important considerations? Or, to put it differently, what important information are we scrubbing away?

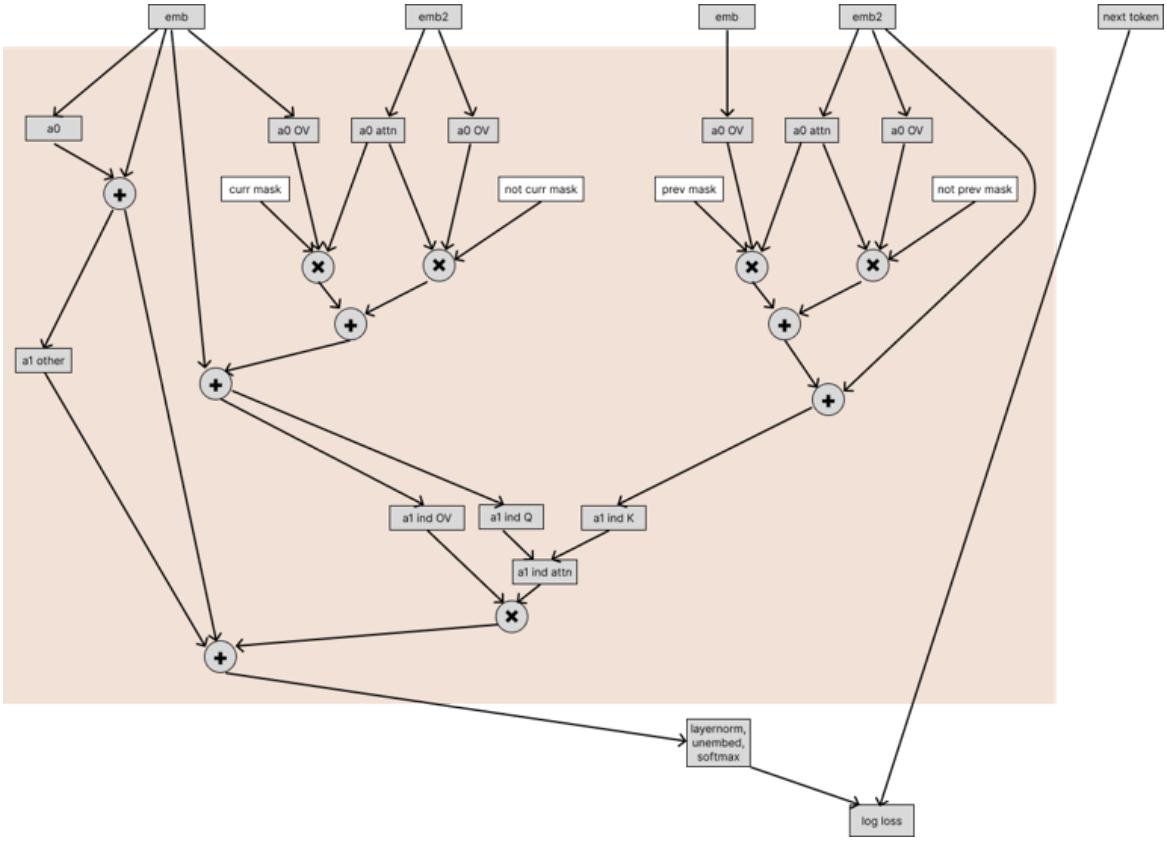
One possibility is that it's common for attention heads to attend substantially to the current sequence position (you'll see this if you look at the attention patterns included in the "Identification" section). This attention results in the token's representation being transformed in a predictable way. And so, when the induction heads are learning to e.g. copy a token value, they'll probably set up their V matrix to take into account the average attention-to-current-token of the layer zero heads.

We would like to express the hypothesis that the induction head interacts with all the layer zero heads, but through their average attention-to-current-token. That is, we hypothesize that the induction head's behavior isn't importantly relying on the ways that a0 heads vary their attention depending on context; it's just relying on the effect of the a0 head OV pathway, ignoring correlation with the a0 attention pattern.

Similarly, there might be attention heads other than the previous token head which, on average, attend substantially to the previous token; the previous hypothesis also neglects this, but we'd like to represent it.

Here's the complete experiment we run. Things to note:

- We've drawn the "emb" and "emb2" nodes multiple times. This is just for ease of drawing–we'll always use the same value the two places we drew an emb node.
- The main point of this experiment is that the layer zero attention patterns used by the induction heads always come from emb2, so the induction heads can't be relying on any statistical relationship between the layer zero attention pattern and the correct next token.



Running parts of this individually (that is, just scrubbing one of Q, K, or V in the induction heads, while giving the others their value on emb) and all together (which is what is pictured) yields the following amounts of loss recovered:

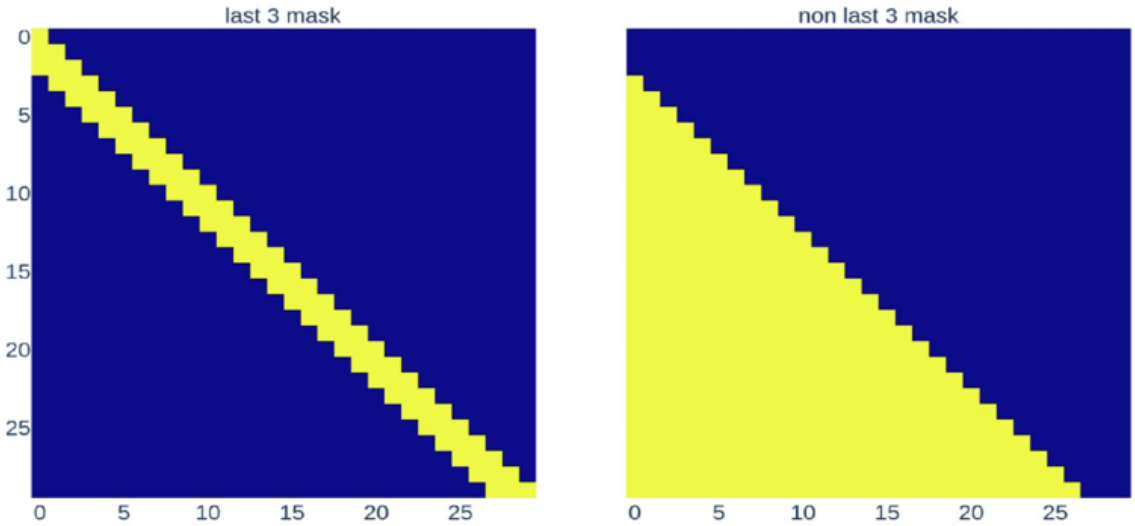
Q: 76%
K: 86%
V: 97%
All: 62%

So, we've captured V quite well with this addition, but we haven’t yet captured much of what’s happening with K and Q.

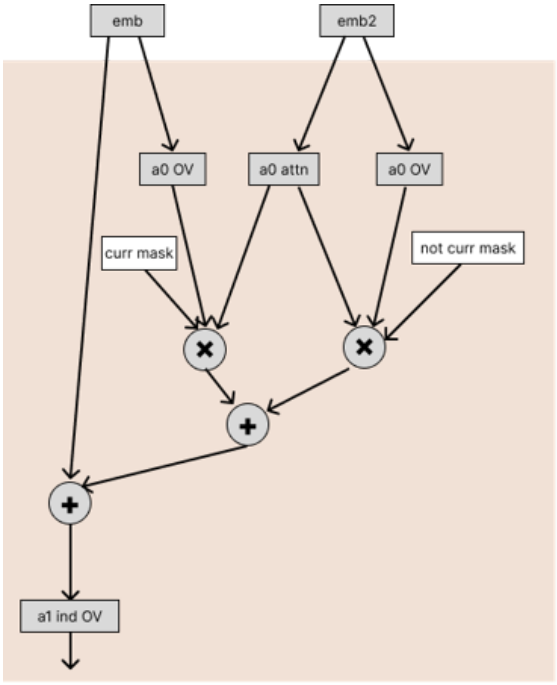
Refined hypothesis 2

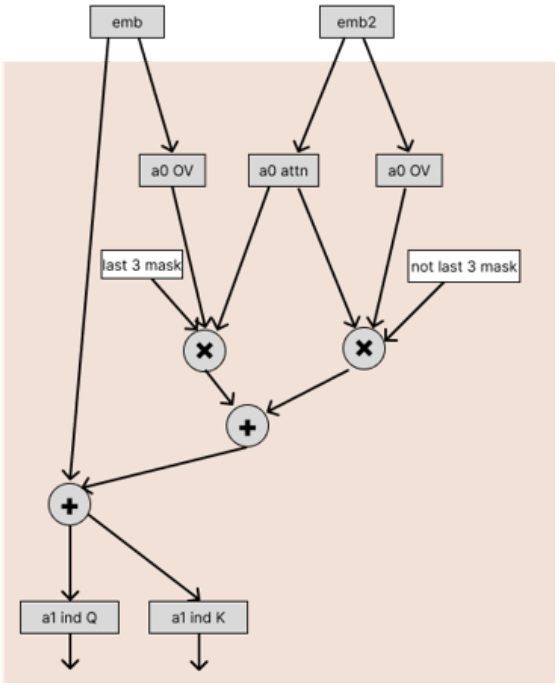
One theory for what could be going wrong with Q and K is that we need to take into account other sequence positions. Specifically, maybe there's some gating where K only inducts on certain 'B' tokens in AB...A, and maybe the induction heads fire harder on patterns of the form XAB...XA, where there are two matching tokens (for example, in the earlier Dursley example, note that the two previous tokens | D| and |urs| both matched.). This is certainly not a novel idea—prior work has mentioned fuzzy matching on multiple tokens.

So, we'll considerably expand our hypothesis by including 'just the last 3 tokens' for K and Q (instead of just previous and just current). (By last three, we mean current, previous, and previous to previous.)

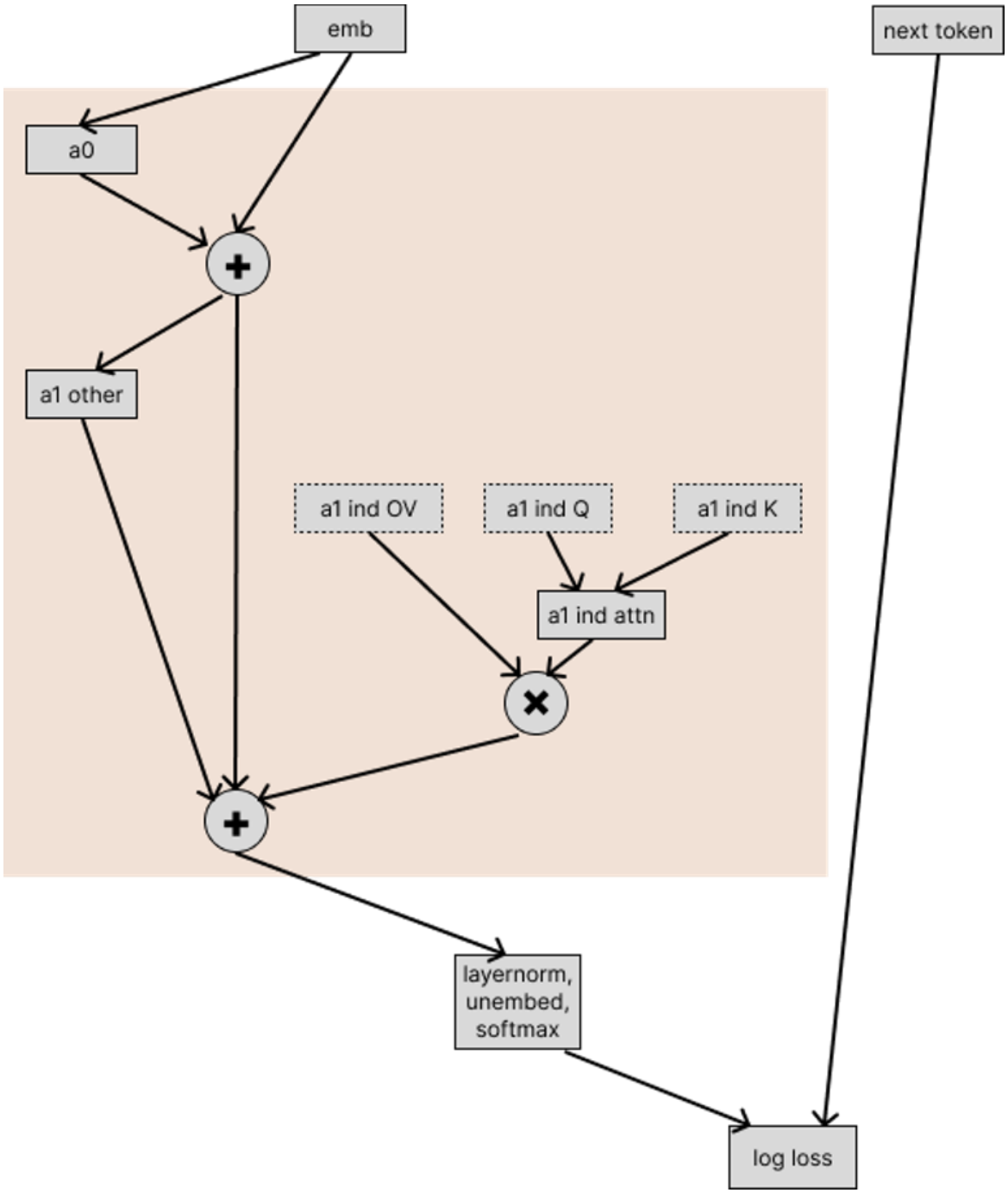


It’s getting unwieldy to put all this in the same diagram, so we’ll separately draw how to scrub K, Q, and V. The OV activations are produced using the current token mask, and the Q and K are produced using the “last 3 mask”. Both use the direct path from emb rather than emb2.





Given these, we can do the experiments for this hypothesis by substituting in those scrubbed activations as desired:



And the numbers are:

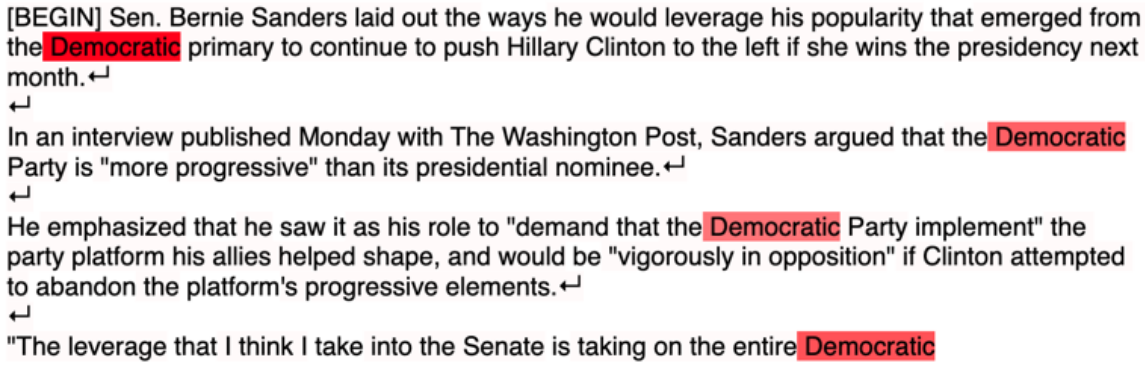
Q: 87%
K: 91%
V: 97% (same as previous)
All: 76%

This improved things considerably, but we're still missing quite a bit. (We tested using different subsets of the relative sequence positions for Q and K; using the last three for both was the minimal subset which captures nearly all of the effect.)

Refined hypothesis 3

If you investigate what heads in layer 0 do, it turns out that there are some heads which often almost entirely attend to occurrences of the current token, even when it occurred at earlier sequence positions.

The below figure shows the attention pattern of 0.2 for the query at the last ' Democratic' token:



Link to interface

So you can see that 0.2 attended to all the copies of " Democratic".

Because this is a layer zero head, the input to the attention head is just the token embedding, and so attending to other copies of the same token leads to the same output as the head would have had if it had just attended to the current token. But it means that on any particular sequence, this head's attention pattern is quite different from its attention pattern averaged over sequences. Here is that head's attention pattern at that sequence position, averaged over a large number of sequences:

On average, this head attends mostly to the current token, a bit to the `[BEGIN]` token, and then diffusely across the whole sequence. This is the average attention pattern because tokens that match the current token are similarly likely to be anywhere in the context.

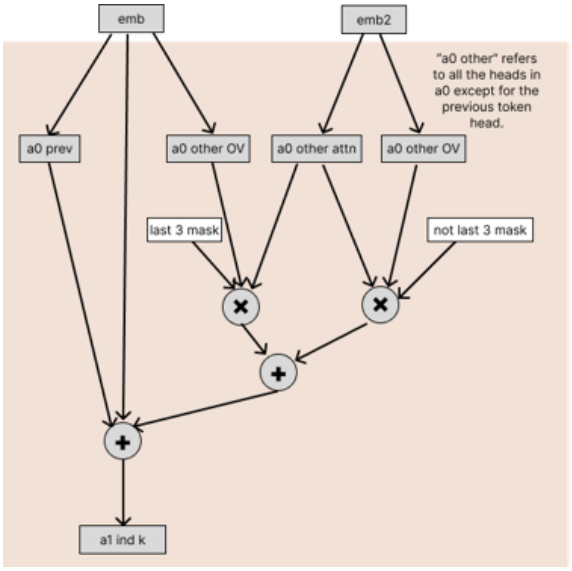
These heads have this kind of attend-to-tokens-that-are-the-same-as-the-current-token behavior for most of the tokens in the subset of tokens that we picked (as described in "Picking out inductiony tokens"). This is problematic for our strategy where we scrub the attention probabilities because the expected attention probability on tokens matching the current token might be 0.3, even though the model always only attends to tokens matching the current token.

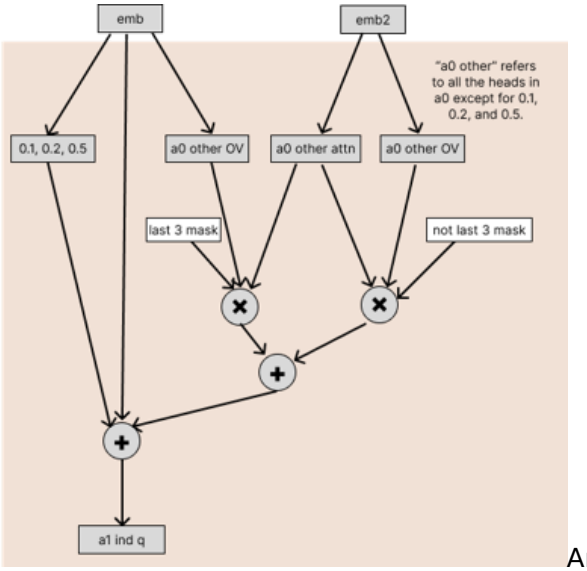
There are two-layer 0 heads which most clearly have this behavior, 0.2 and 0.5, as well as 0.1, which somewhat has this behavior.

(These heads don't *just* do this. For instance, in the attention pattern displayed above, 0.2 also attends to ' Democratic' and ' Party' from the ' GOP' token. We hypothesize this is related to 'soft induction'[3], though it probably also has other purposes – for instance directly making predictions from bigrams and usages in other layer 1 heads.)

In addition to this issue with the self-attending heads, the previous token head also sometimes deviates from attending to the previous token, and this causes additional noise when we try to approximate it by its expectation. So, let's try the experiment where we run the previous token head and these self-attending heads with no scrubbing or masking.

So we're computing the queries and keys for the induction heads as follows:





And then we use these for the queries and keys for the induction heads. We use the same values for the induction heads as in the last experiment. Our experiment graph is the same as the last experiment, except that we've produced Q and K for the induction heads in this new way.

Now we get:

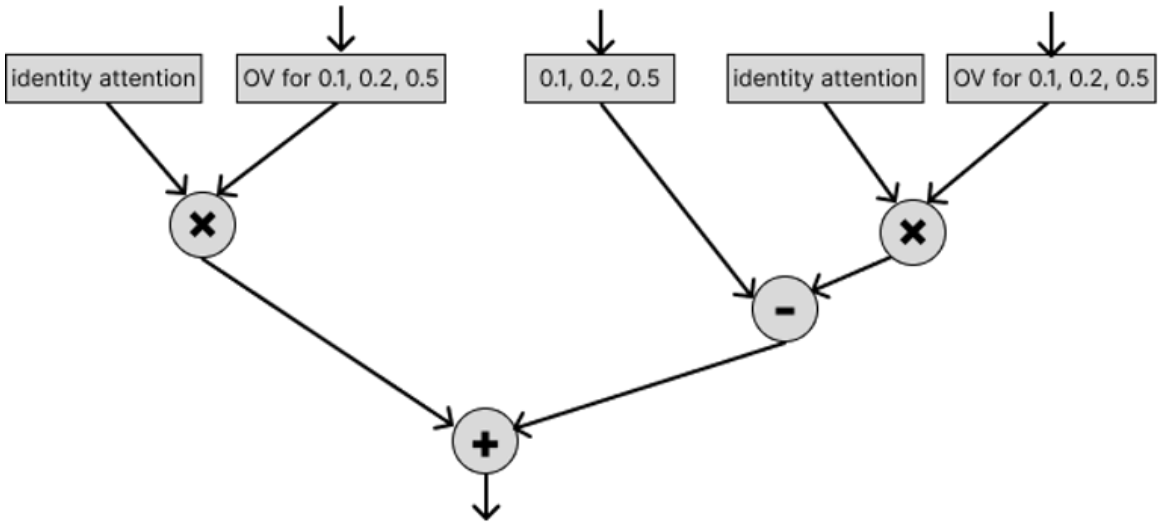
Q: 98%
K: 97%
V: 97% (same as previous)
All: 91%

We're happy with recovering this much of the loss, but we aren't happy with the specificity of our hypothesis (in the sense that a more specific hypothesis makes more mechanistic claims and permits more extreme intervention experiments). Next, we'll try to find a hypothesis that is more specific while recovering a similar amount of the loss.

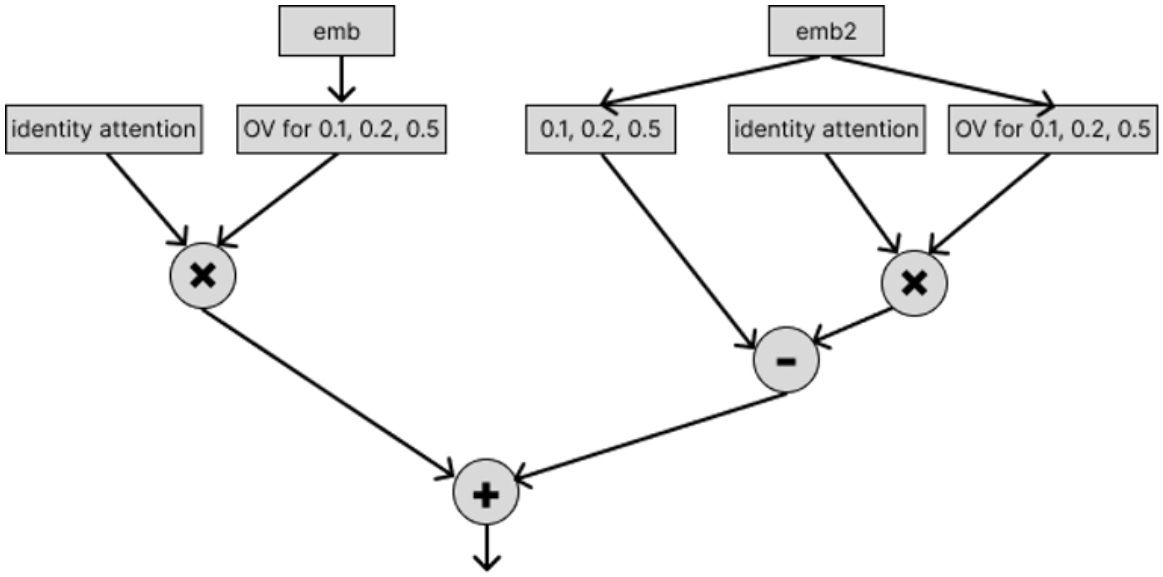
Refined hypothesis 4

So we've observed that the self-attending heads in layer zero are mostly just attending to copies of the same token. This means that even though these heads don't have an attention pattern that looks like the identity matrix, they should behave very similarly to how they'd behave if their attention pattern *was* the identity matrix. If we can take that into account, we should be able to capture more of how the queries are formed.

To test that, we rewrite the self-attending heads (0.1, 0.2, 0.5) using the $a = b + (a - b)$ identity, where "identity attention" means the identity matrix attention pattern:

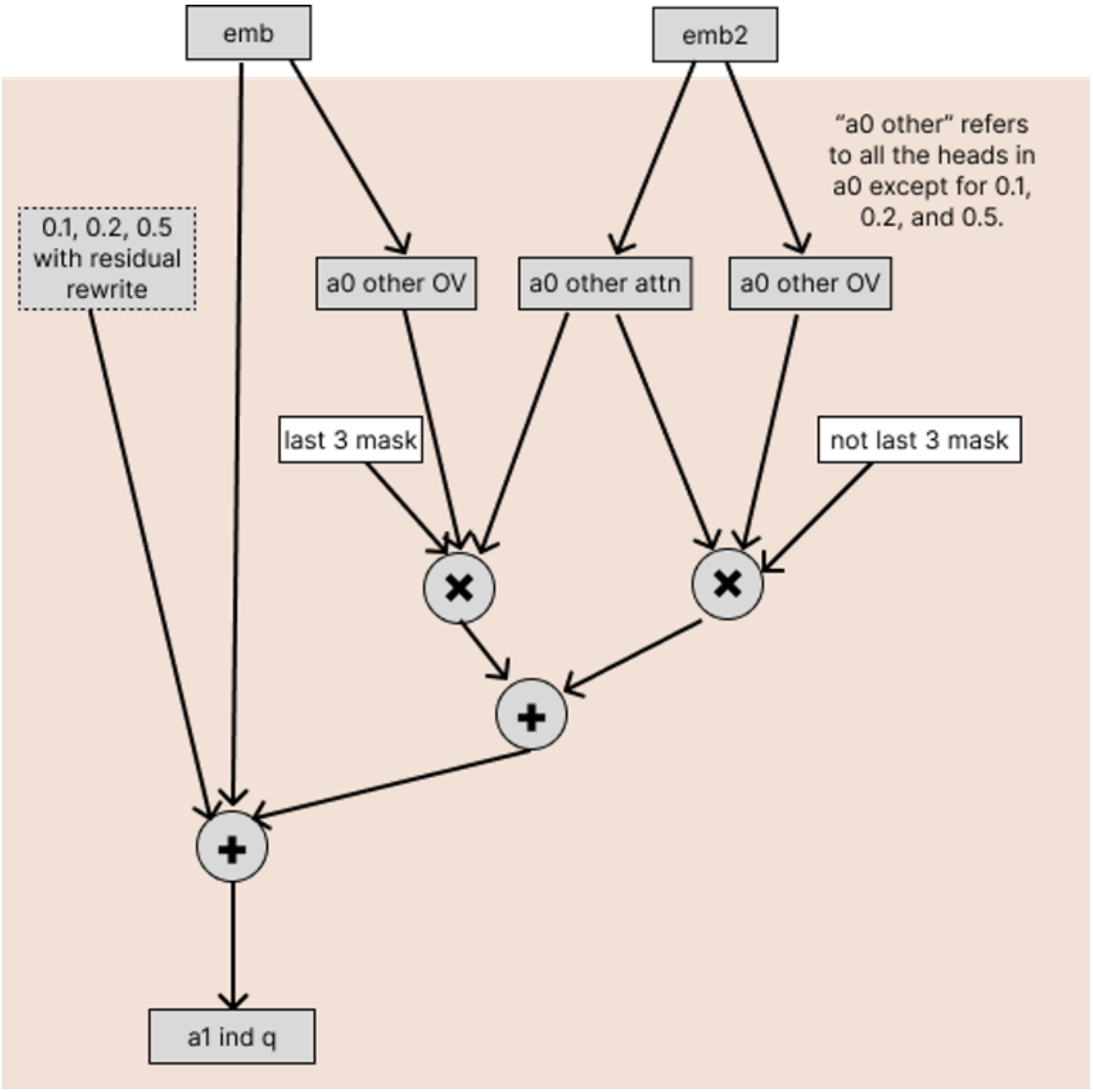


This is equal to calculating 0.1, 0.2, and 0.5 the normal way, but it permits us to check the claim “The outputs of 0.1, 0.2, and 0.5 don’t importantly differ from what they’d be if they always attended to the current token”, by using just the left hand side from the real input and calculating the “error term” using the other input.



Let’s call this “0.1, 0.2, 0.5 with residual rewrite”.

So now we have a new way of calculating the queries for the induction heads:



Aside from the queries for the induction heads, we run the same experiment as refined hypothesis 2.

And this yields:

Q: 97%
K: 91% (same as refined hypothesis 2)
V: 97% (same as refined hypothesis 2)
All: 86%

We've now retained a decent fraction of the loss while simultaneously testing a reasonably specific hypothesis for what is going on in layer 0.

Conclusion

In this post, we used causal scrubbing to test a naive hypothesis about induction heads, and found that it was incorrect. We then iteratively refined four hypotheses using the scrubbed

expectation as a guide. Hopefully, this will serve as a useful example of how causal scrubbing works in simple settings.

Key takeaways:

- We were able to use causal scrubbing to narrow down what model computations are importantly involved in induction.
- In practice, induction heads in small models take into account information from a variety of sources to determine where to attend.

Appendices

Model architecture and experiment details

- The model uses the shortformer positional encodings (which means that the positional embeddings are added into the Q and K values before every attention pattern is computed, but are not provided to the V)
- The model has layer norms before the attention layers
- The model was trained on the openwebtext dataset
- Its hidden dimension is 256
- We ran causal scrubbing on validation data from openwebtext with sequence length 300

How we picked the subset of tokens

Choose beta and threshold. Then for all sequential pairs of tokens, AB, in the corpus we compute:

- The log loss of the bigram probabilities P_{bigram} (via the full bigram matrix).
- The log loss of the *beta-level induction heuristic* probabilities $P_{\beta\text{-induction}}$. Intuitively, this heuristic upweights the probability of B based on how frequently A has been followed by B in the context. We compute these probabilities as:
  - Find all prior occurrences of A in the same input datum
  - Count the number of these prior occurrences which are followed by B. Call this the matching count m. Let the remaining occurrences be the not matching count, n.
  - Starting from the bigram statistics, we add $\beta * m$ to the logit of B and $\beta * n$ to the not-B logit. That is:

    $$P_{\beta\text{-induction}}(B|A) = \sigma(\log P_{bigram}(B|A) + \beta \cdot m - \log P_{bigram}(\neg B|A)) - \beta \cdot n)$$
  - Then, we compute the log loss of these probabilities.
- Finally, we compute the average log loss from the A for each of these two heuristics. If the bigram loss is larger than the induction heuristic loss by at least some small threshold for a given A token, we include that token.

(We can also run the same experiments while evaluating the loss on *all* tokens; this decreases the loss recovered relative to evaluating on just this subset.)

Bonus refined hypothesis 5

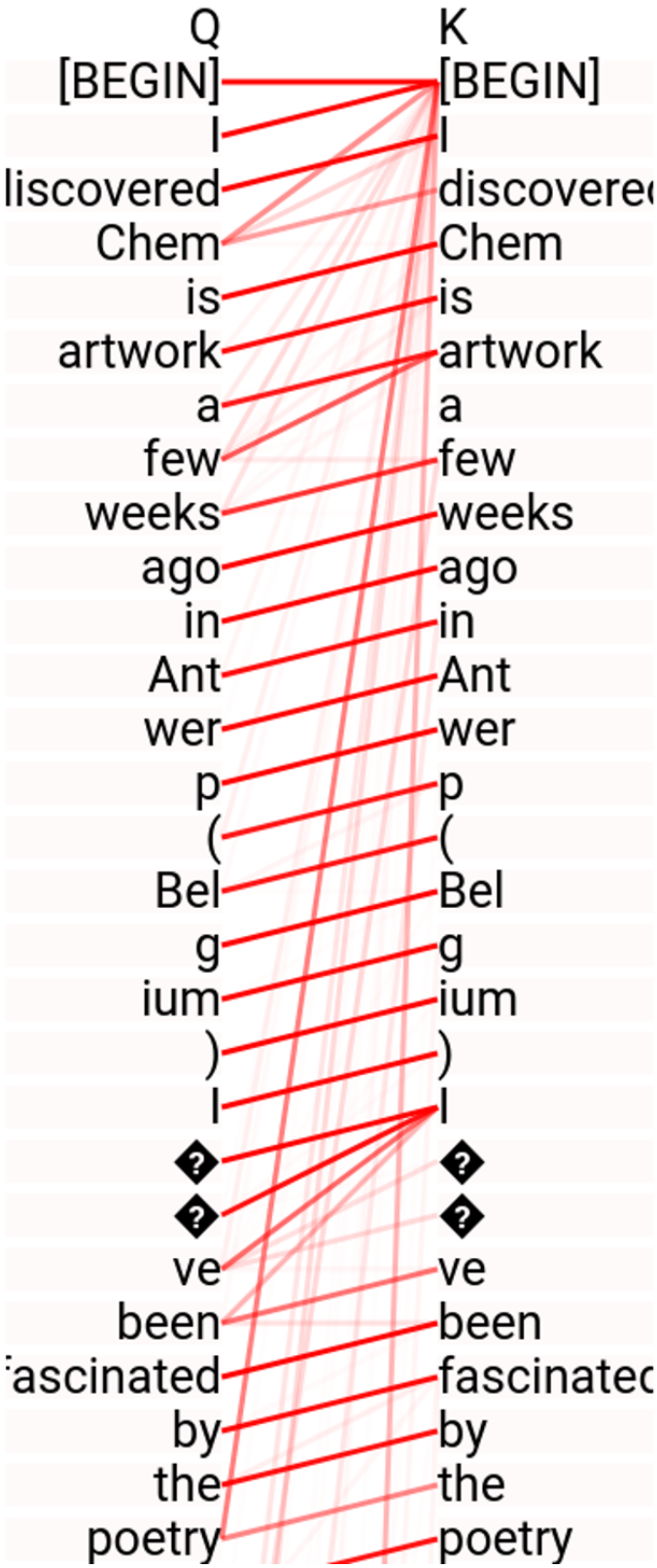
Our previous hypothesis improved the Q pathway considerably, but we're missing quite a bit of loss due to scrubbing the attention pattern of the previous token head for K. This is due to cases where the previous token head deviates from attending to the previous token. If we sample an alternative sequence which fails to attend to the previous token at some important location, this degrades the induction loss. We'll propose a simple hypothesis which partially addresses this issue.

Consider the following passage of text:

```
            [BEGIN] I discovered Chemis artwork a few weeks ago in Antwerp (Belgium) I've
been fascinated by the poetry of his murals and also by his fantastic technique. He kindly
accepted to answer a few questions for StreetArt360. Here's his interview.\n\nHello
Dmitrij, great to meet you.
```

We'll look at the attention pattern for the previous token head as weighted lines from q tokens to k tokens.

For instance, here's the start of the sequence:



[Link to interactive interface](#)

Some utf-8 characters consist of multiple bytes and the bytes are tokenized into different tokens. In this case, those black diamonds are from an apostrophe being tokenized into 2

tokens.

Note that the previous token head exhibits quite different behavior on `I’ve` (where the apostrophe is tokenized into those 2 black diamonds). Specifically, `ve` skips over the apostrophe to attend to `I`. The token `been` also does this to some extent.

Another common case where the previous token head deviates is on punctuation. Specifically, on periods it will often attend to the current token more than typical.

While there are a variety of different contexts in which the previous token head importantly varies its attention, we’ll just try to explain a bit of this behavior. We’ll do this by identifying a bunch of cases where the head has atypical behavior.

It turns out that the model special cases a variety of different quote characters. We’ll specifically reference `'`, “, ”, and ”.

It’s a bit hard to read this, so here’s a zoomed in version with a font that makes the different characters more obvious.

These quote characters each consist of two tokens and the head has atypical behavior if any of those tokens is the previous one.

It turns out the model also sometimes has atypical behavior if the previous token is the `a` or `an` token:

And, it has atypical behavior if the current token is `.` or `to`.

Overall, here is simple classifier for whether or not a given token is atypical:

- Is the previous token one of the bytes from any of: ’, “, ”, and ”? If so, atypical.
- Is the previous token one of `a` or `an`? If so, atypical.
- Is the current token one of `.` or `to`? If so, atypical.
- Otherwise, typical.

And we’ll propose that it only matters whether or not the attention from the current location should be ‘typical’ or ‘atypical’. Then, we can test this hypothesis with causal scrubbing by sampling the attention pattern from the current location from a different sequence which has the same ‘typicality’ at that location.

This hypothesis is clearly somewhat wrong – it doesn’t only matter whether or not the current token is ‘typical’! For instance, the current token being `.` or `to` results in attending to the previous token while the previous token being `a` results in attending two back. Beyond this issue, we’ve failed to handle a bunch of cases where the model has different behavior. This hypothesis would be easy to improve, but we’ll keep it as is for simplicity.

We’ll apply this augmented hypothesis for the previous token head to just the K pathway. This yields:

Q: 97% (same as refined hypothesis 4)
K: 94%
V: 97% (same as refined hypothesis 2)
All: 89%

So compared to refined hypothesis 4, this improved the loss recovered by 3% for both K individually (94 -> 97%) and everything together (86->89%). This brings us considerably closer to the overall loss from hypothesis 3 which was 91%.

1. ^

   These heads are probably doing some things in addition to induction; we'll nevertheless refer to them as induction heads for simplicity and consistency with earlier work.

2. ^

   Note that we opt to use a single input rather than many.

3. ^

   This is where seeing AB updates you towards thinking that tokens similar to A are likely to be followed by tokens like B.

Causal scrubbing: results on a paren balance checker

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

* *Authors sorted alphabetically.*

This is a more detailed look at our work applying causal scrubbing to an algorithmic model. The results are also summarized here .

Introduction

In earlier work (unpublished), we dissected a tiny transformer that classifies whether a string of parentheses is balanced or unbalanced.[1] We hypothesized the functions of various parts of the model and how they combine to solve the classification task. The result of this work was a qualitative explanation of how this model works, but one that made falsifiable predictions and thus qualified as an informal hypothesis. We summarize this explanation below.

We found that the high-level claims in this informal hypothesis held up well (88-93% loss recovered, see the Methodology). Some more detailed claims about how the model represents information did not hold up as well (72%), indicating there are still important pieces of the model's behavior we have not explained. See the experiments summary section for an explanation of each hypothesis refinement.

Causal scrubbing provides a language for expressing explanations in a formal way. A formal hypothesis is an account of the information present at every part of the model, how this information is combined to produce the output, and (optionally) how this information is represented. In this work, we start by testing a simple explanation, then iterate on our hypothesis either by improving its accuracy (which features of the input are used in the model) or specificity (what parts of the model compute which features, and optionally how). This iterative process is guided by the informal hypothesis that we established in prior work.

For a given formal hypothesis, the causal scrubbing algorithm automatically determines the set of interventions to the model that would not disturb the computation specified by the hypothesis. We then apply a random selection of these interventions and compare the performance to that of the original model.

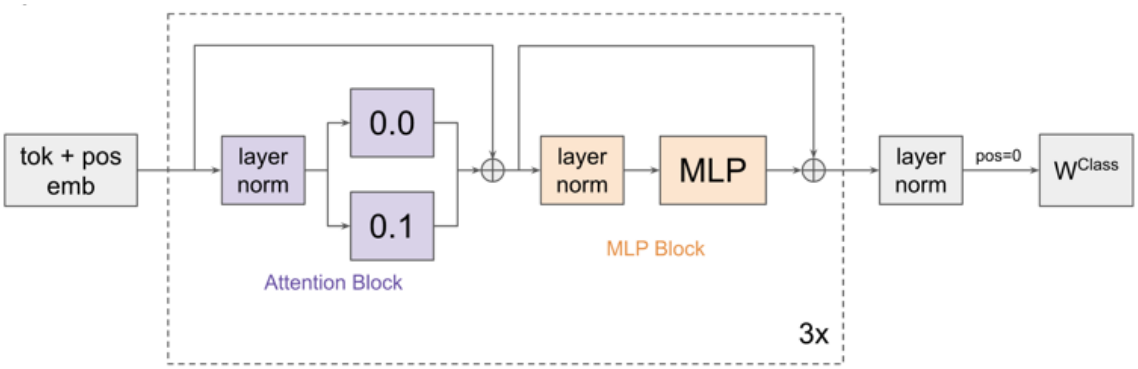
Using causal scrubbing to evaluate our hypotheses enabled us to identify gaps in our understanding, and provided trustworthy evidence about whether we filled those gaps. The concrete feedback from a quantitative correctness measure allowed us to focus on quickly developing alternative hypotheses, and finely distinguish between explanations with subtle differences.

We hope this walk-through will be useful for anyone interested in developing and evaluating hypothesized explanations of model behaviors.

Setup

Model and dataset

The model architecture is a three layer transformer with two attention heads and pre layernorm:



There is no causal mask in the attention (bidirectional attention). The model is trained to classify sequences of up to 40 parentheses. Shorter sequences are padded, and the padding tokens are masked so they cannot be attended to.

The training data set consists of 100k sequences along with labels indicating whether the sequence is balanced. An example input sequence is `()())()` which is labeled *unbalanced*. The dataset is a mixture of randomly generated sequences and adversarial examples.[2] We prepend a `[BEGIN]` token at the start of each sequence, and read off the classification above this token (therefore, the parentheses start at sequence position 1).

For the experiments in this writeup we only use random, non-adversarial, inputs, on which the model is almost perfect (loss of 0.0003, accuracy 99.99%). For more details of the dataset see the appendix.

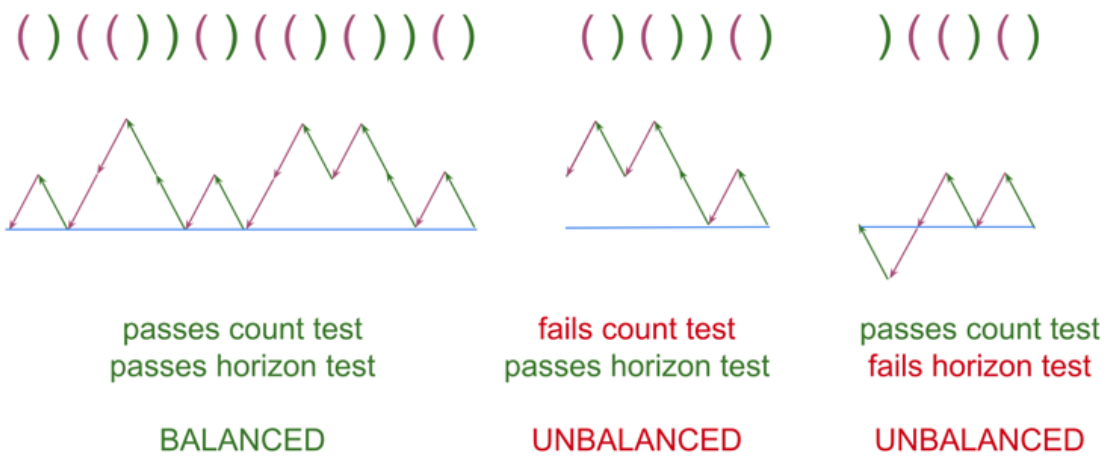
Algorithm

Our hypothesis is that the model is implementing an algorithm that is approximately as follows:

1. Scan the sequence from right to left and track the nesting depth at each position. That is, the nesting depth starts at 0 and then, as we move across the sequence, increments at each `)` and decrements at each `(`.

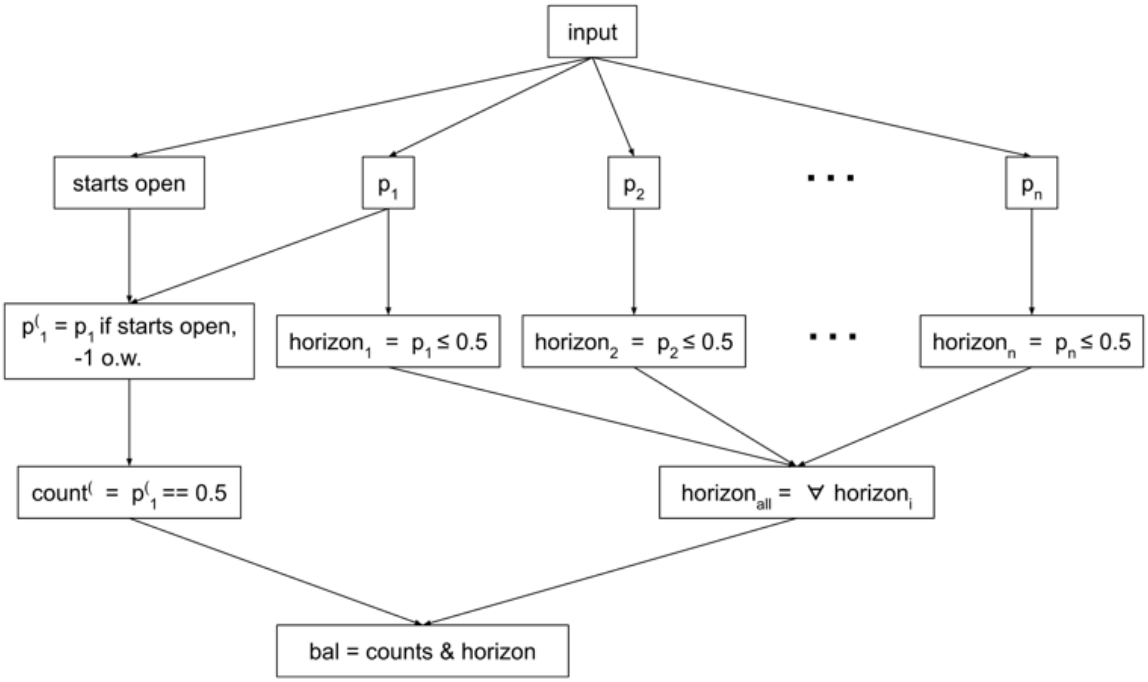
   You can think of this as an "elevation profile" of the nesting level across the sequence, which rises or falls according to what parenthesis is encountered.

   Important note: scanning from left to right and scanning from right to left are obviously equally effective. The specific model we investigated scans from right to left (likely because we read off the classification at position 0).
2. Check two conditions:
   1. **The Equal Count Test (aka the count test)**: Is the elevation back to 0 at the left-most-parentheses? This is equivalent to checking whether there are the same number of open and close parentheses in the entire sequence.
   2. **The Above Horizon Test (aka the horizon test)**: Is the elevation non-negative at every position? This is equivalent to checking whether there is at least one open parenthesis `(` that has not been later closed `)` (cf. see the third example below).
3. If either test fails, the sequence is unbalanced. If both pass, the sequence is balanced.



(As an aside, this is a natural algorithm; it's also similar to what codex or GPT-3 generate when prompted to balance parentheses.)

Again - this is close to the algorithm we hypothesize the model uses, but not exactly the same. The algorithm that the model implements, according to our hypothesis, has two differences from the one just described.



1. It uses proportions instead of 'elevation'. Instead of computing 'elevation' by incrementing and decrementing a counter, we believe the model tracks the proportion of parentheses that are open in each suffix-substring (i.e. in every substring that ends with the rightmost parenthesis). This proportion contains the same information as 'elevation'. We define:

p_i: the proportion of open parentheses (in the suffix-substring starting at position i, i.e. from i to the rightmost parenthesis

Put in terms of proportions, the Equal Count Test is whether this is exactly 0.5 for the entire string ($p_1 == 0.5$). The Not Beneath Horizon Test is whether this is less than or equal to 0.5 for each suffix-substring ($p_i <= 0.5$ for all i); if the proportion is less than 0.5 at any point, this test is failed.

2. It uses a combined test of, "is the first parenthesis open, and does the sequence pass Equal Count?" Call this the Start-Open-and-Equal-Count test, aka the count$^{(}$ Test.

Consider the Start-Open component. Sequences that start with a closed parenthesis instead of an open one cannot be balanced: they inevitably fail to meet *at least* one of Equal Count or Not Beneath Horizon. However, the model actually computes Start-Open separately!

As the model detects Start-Open in the circuit that computes the 'equal count' test, we've lumped them together for cleaner notation:

> **count$^{(}$**: Is the elevation back to 0 at the left-most-parenthesis (i.e. the Equal Count Test), and does the sequence start with open parenthesis? i.e. count$^{(}$:= (first parenthesis is open) & (passes Equal Count Test)

We can use these variables to define a computational graph that will compute if any sequence of n parentheses is balanced or not.

Note that we define count$^{(}$, horizon$_i$, and horizon$_{all}$ to be booleans that are true if the test passes (implying that the sequence might be balanced) and false (implying the sequence definitely isn't balanced).

We will reference the features of the above graph to make claims about what particular components of the model compute.

Informal Hypothesis

Our prior interpretability work suggested that the model implements this algorithm in roughly the following manner:

- Head 0.0 has an upper triangular attention pattern (recall that model uses bidirectional attention): at every query position it pays roughly-equal attention to all following sequence positions and writes in opposite directions at open and close parentheses. These opposite directions are analogous to "up" and "down" in the elevation profile. Thus, head 0.0 computes every p_i and writes this in a specific direction.
- The MLPs in layers 0 and 1 then transform the p_i into binary features. In particular, at position 1 they compute the count$^{(}$ test, and at every sequence position they compute the horizon test for that position.
- Head 1.0 and 2.0 both copy the information representing the count$^{(}$ test from position 1 (the first parentheses token) to position 0 (the [BEGIN] token where the classifier reads from).
- Head 2.1 checks that the horizon test passed at all positions and writes this to position 0.

A consequence of this hypothesis is that at position 0, head 2.0 classifies a sequence as balanced if it passes the count$^{(}$ test, and classifies it as unbalanced if it fails the test. Head 2.1 does the same for the horizon test. As some evidence for this hypothesis, let's look at an attribution experiment.

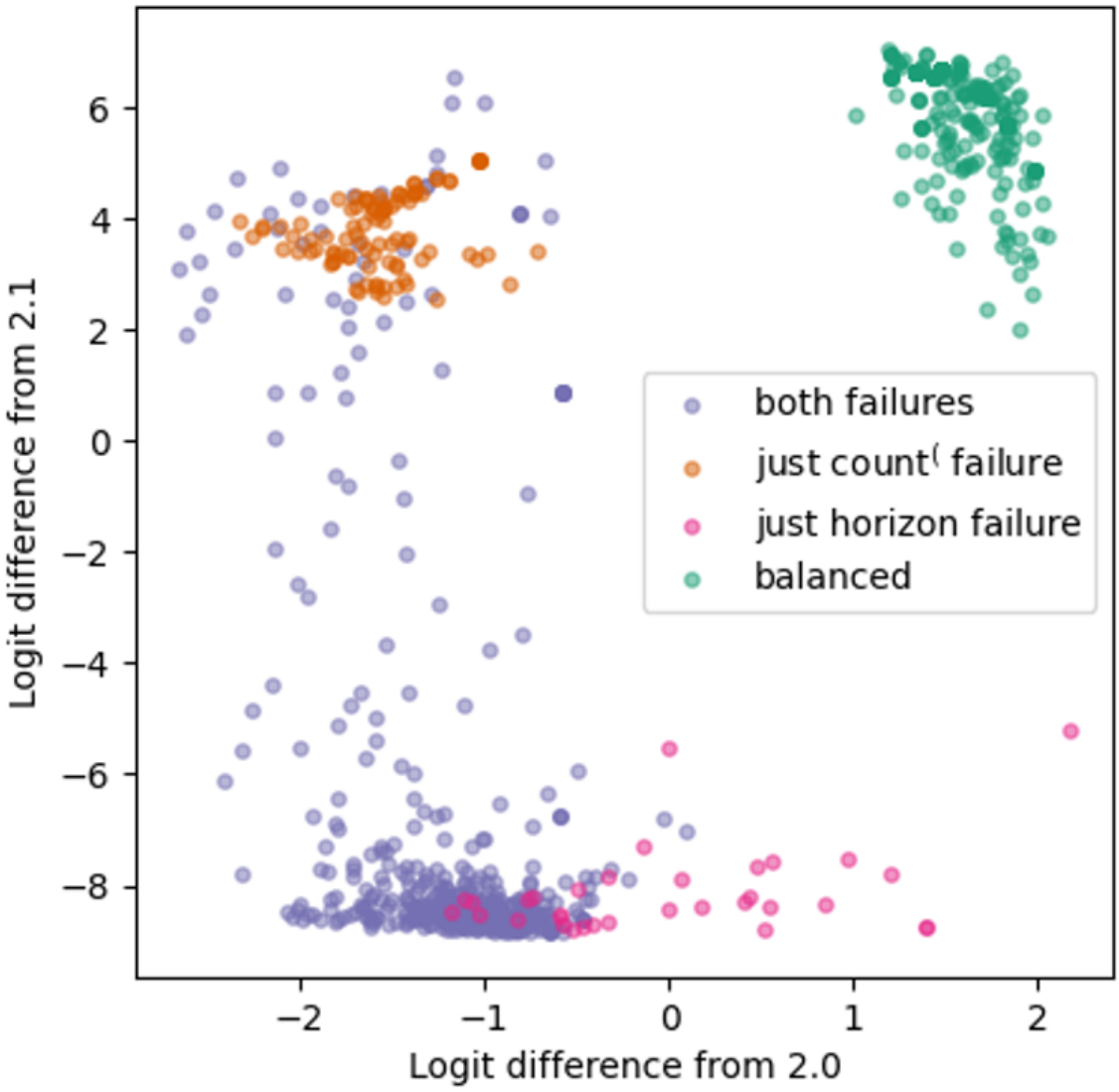
We run the model on points sampled on the random data set, which may each pass or fail either or both of the tests. We can measure the predicted influence on the logits from the

output of heads 2.0 and 2.1.[3]

For each data point, we plot these two values in the x and y axes. If each head independently and perfectly performs its respective test, we should expect to see four clusters of data points:

- Those that pass both tests (i.e. are balanced) are in the top right: both heads classify them as balanced, so their x and y positions are positive.
- Unbalanced sequences, which fail both tests, are points in the bottom left.
- Sequences that pass only one of the tests should be in the top left or bottom right of the plot.

This is the actual result:



The result roughly matches what we expected, but not entirely.

The part that matches our expectations: the green (balanced) points are consistently classified as balanced by the two heads, and the orange (count(failure only) points are consistently classified as balanced by 2.1 and unbalanced by 2.0.

However, the picture for the other clusters does not match our expectations; this shows that our hypothesis is flawed or incomplete. The pink points fail only the horizon test, and should be incorrectly classified as balanced by 2.0, and correctly classified as unbalanced by 2.1. In reality, 2.0 often 'knows' that these sequences are unbalanced, as evidenced by about half of these points being in the negative x axis. It must therefore be doing something other than the count(test, which these sequences pass. The purple points, which fail both the count(and horizon tests, are sometimes incorrectly thought to be balanced by 2.1, so head 2.1 cannot be perfectly performing the horizon test. In Experiment 3, we'll show that causal scrubbing can help us detect that this explanation is flawed, and then derive a more nuanced explanation of head 2.1's behavior.

Methodology

We use the causal scrubbing algorithm in our experiments. To understand this algorithm, we advise reading the introduction post. Other posts are not necessary to understand this post, as we'll be talking through the particular application to our experiments in detail.[4]

Following the causal scrubbing algorithm, we rewrite our model, which is a computational DAG, into a tree that does the same computation when provided with multiple copies of the input. We refer to the rewritten model as the *treeified model*. We perform this rewrite so we can provide separate inputs to different parts of the model–say, a reference input to the branch of the model we say is important, and a random input to the branch we say is unimportant. We'll select sets of inputs randomly conditional on them representing the same information, according to our hypothesis $h = (G, I, c)$ (see the experiments for how we do this), run the treeified model on these inputs, and observe the loss. We call the treeified model with the separate inputs assigned according to the hypothesis the "scrubbed model".

Before anything else, we record the loss of the model under two trivial hypotheses: "everything matters" and "nothing matters". If a hypothesis we propose is perfect, we expect that the performance of the scrubbed model is equal to that of the baseline, unperturbed model. If the information the hypothesis specifies is unrelated to how the model works, we expect the model's performance to go down to randomly guessing. Most hypotheses we consider are somewhere in the middle, and we express this as a *% loss recovered* between the two extremes. For more information on this metric, refer to the relevant section here.

Summary of experimental results

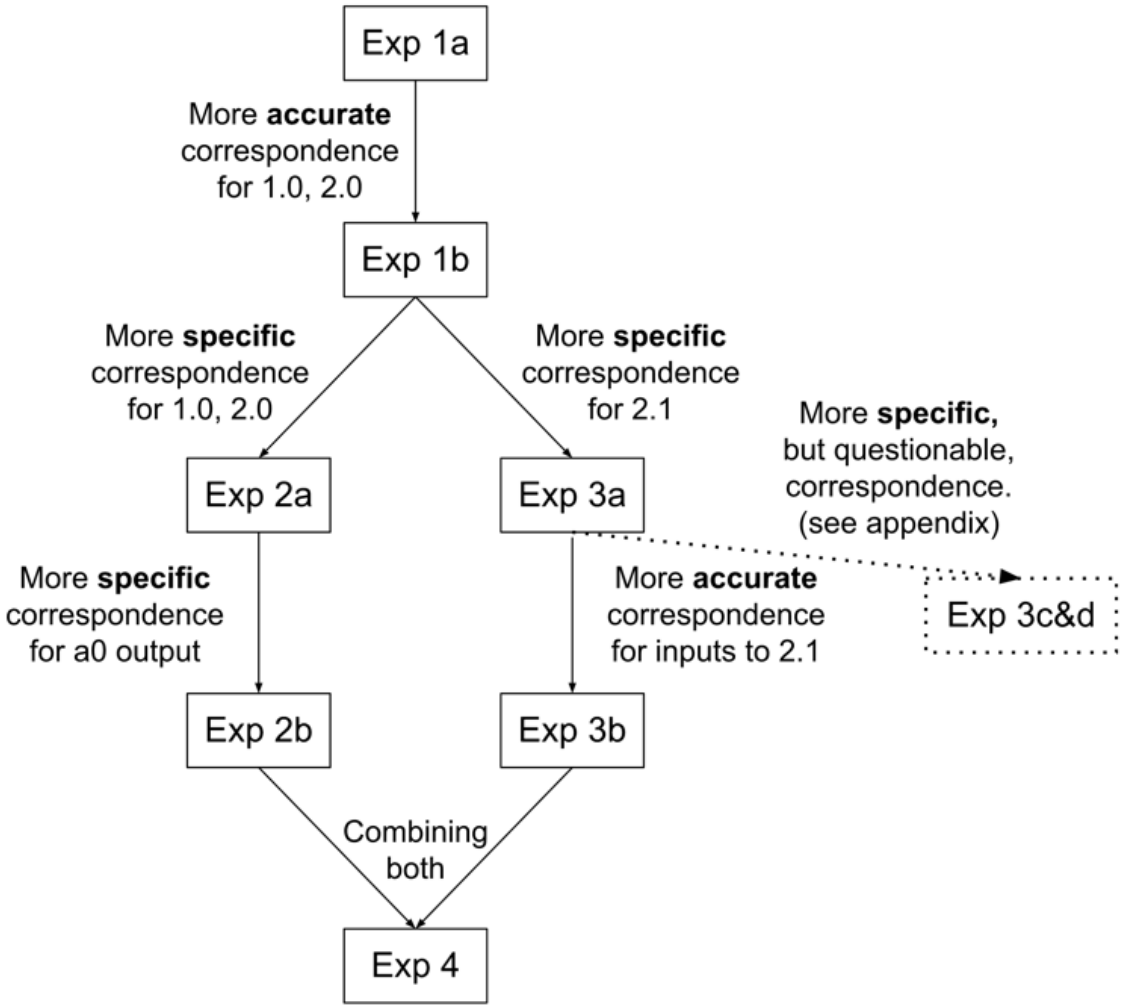
We run a series of experiments to test different formalizations of (parts of) the informal hypothesis we have about this model.

We start with a basic claim about our model: that there are only three heads whose direct contribution to the classifier head is important: 1.0 and 2.0 which compute count(, and 2.1 which computes the horizon test. We then improve this claim in two ways:

- **Specificity:** Making a more *specific* claim about how one of these pathways computes the relevant test. That is, we claim a more narrow set of features of the input are important, and therefore *increase* the set of allowed interventions. This is necessary if we want to claim to understand the complete computation of the model, from inputs to outputs. However, it generally increases the loss of the scrubbed model if the additions to the hypothesis are imperfect.
- **Accuracy:** *Improving* our hypothesis to more accurately match what the model computes. This often involves adjusting the features computed by our interpretation I. If done correctly this should decrease the loss of the scrubbed model.

A third way to iterate the hypothesis would be to make it more *comprehensive*, either by including paths through the model that were previously claimed to be unimportant or by being more restrictive in the swaps allowed for a particular intermediate. This should generally decrease the loss. We don't highlight this type of improvement in the document, although it was a part of the research process as we discovered which pathways were necessary to include in our explanation.

Our experiments build upon one another in the following way:



The results of the experiments are summarized in this table, which may be helpful to refer back to as we discuss the experiments individually.

#	Summary of claimed hypothesis	Loss ± Std. Error	% loss recovered	Accuracy
0a	The normal, unscrubbed, model	0.0003	100%	100%
0b	Randomized baseline	4.30 ± 0.12	0%	61%

1a	1.0 and 2.0 compute the count test, 2.1 computes the horizon test, they are ANDed	0.52 ± 0.04	88%	88%
1b	1a but using the count(test	0.30 ± 0.03	93%	91%
2a	More specific version of 1b, where we specify the inputs to 1.0 and 2.0	0.55 ± 0.04	88%	87%
2b	2a but using the ϕ approximation for the output of 0.0	0.53 ± 0.04	88%	87%
3a	More specific version of 1b, where we break up the inputs to 2.1 by sequence position	0.97 ± 0.06	77%	85%
3b	3a but using p_{adj}	0.68 ± 0.05	84%	88%
3c	3a plus specifying the inputs to 2.1 at each sequence position	0.69 ± 0.05	84%	87%
3d	3a but sampling a1 at each sequence position randomly	0.81 ± 0.05	81%	87%
4	Including both 2b and 3b	1.22 ± 0.07	72%	82%

(% loss recovered is defined to be 1 - (experiment loss - 0a loss) / 0b loss. This normalizes the loss to be between 0% and 100%, where higher numbers are better.)

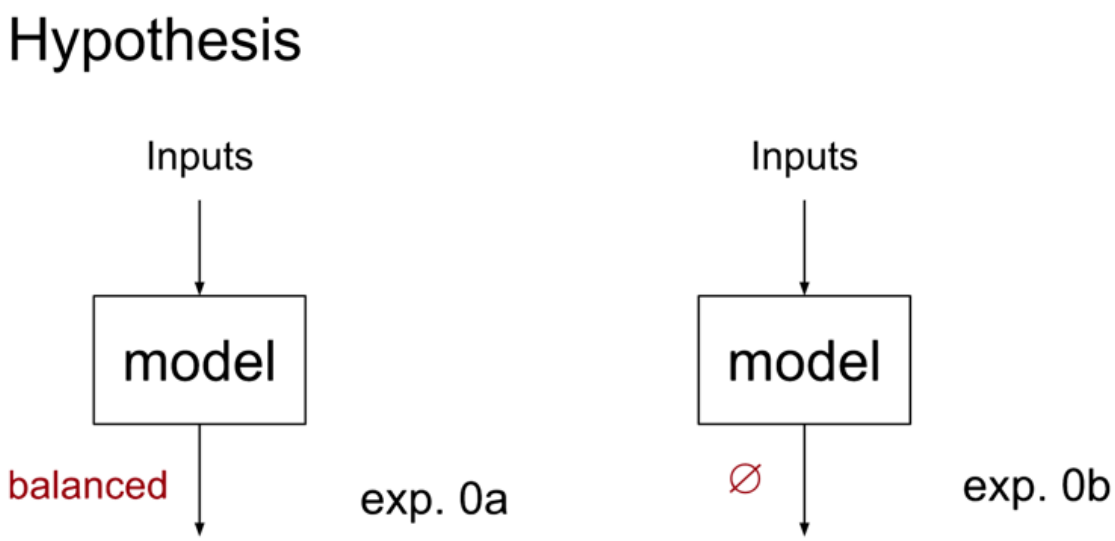
All experiments are run on 2000 scrubbed inputs, sampled according to the algorithm from 100,000 sequences of parentheses.

Detailed experimental results

Experiment 0: Trivial hypothesis baseline

Running the model itself results in a loss of 0.0003 (100% accuracy) on this dataset. If you shuffle the labels randomly, this results in a loss of 4.30 (61% accuracy – recall the dataset is mostly unbalanced).

These can both be formalized as trivial hypotheses, as depicted in the diagram below. We hypothesize an interpretation I with a single node, which corresponds (via c) to the entire model. The computational graph of I, labeled with the model component it corresponds to, is shown below in black. The proposed feature computed by the node of I is annotated in red.



The toy hypothesis used in experiment 0, where we check that shuffling the model outputs across inputs does indeed harm performance. Note that the performance metric (log loss) is left implicit in this figure, and does not have a corresponding node. As the model only has a single path, we don't need to treeify it.

Note that in both cases we don't split up our model into paths (aka 'treeify' it), meaning we will not perform any internal swaps.

In experiment 0a, we claim the output of the entire model encodes information about whether a given sequence is balanced. This means that we can swap the output of the model only if the label agrees: that is, the output on one balanced sequence for another balanced sequence. This will of course give the same loss as running the model on the dataset.

For 0b, we no longer claim *any* correspondence for this output. We thus swap the outputs randomly among the dataset. This is equivalent to shuffling the labels before evaluating the loss. We call such nodes (where any swap of their output is permitted) 'unimportant' and generally don't include them in correspondence diagrams.

These experiments are useful baselines, and are used to calculate the % loss recovered metric.

Experiment 1: Contributions to residual stream at pos0

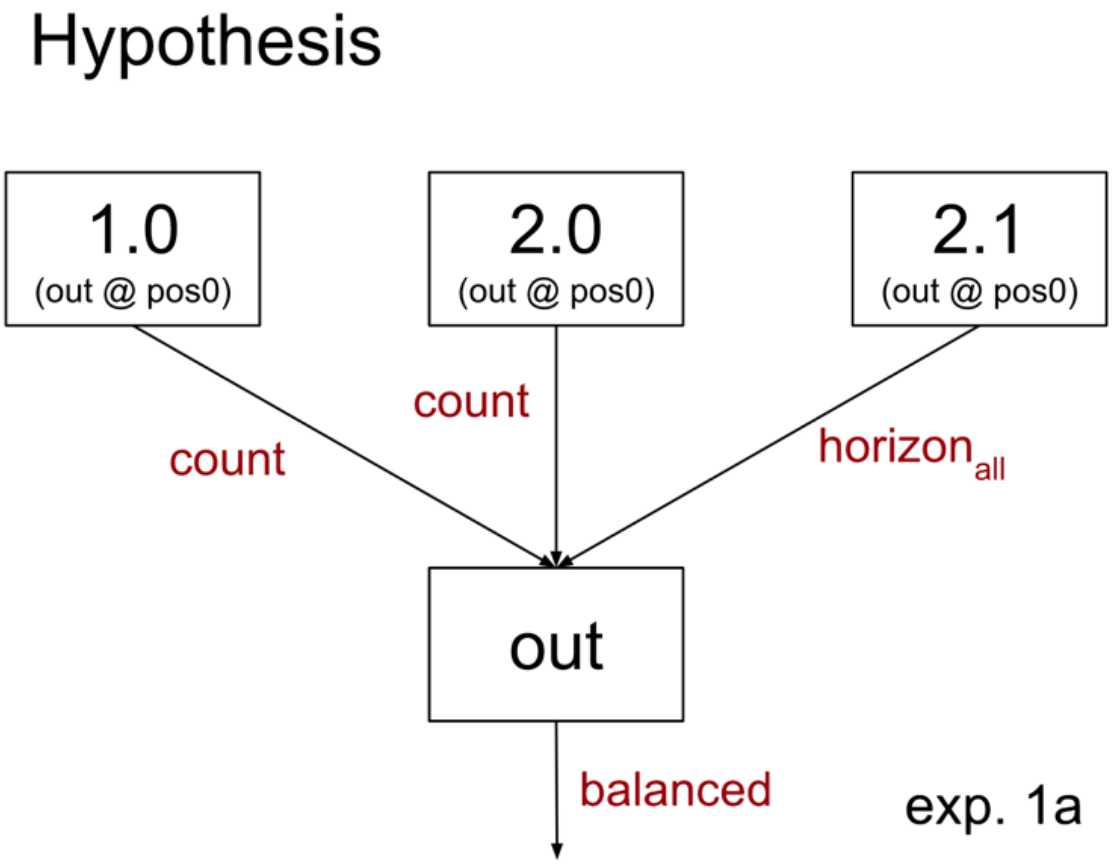
We claimed that the output of 1.0 and 2.0 each correspond to the count(test, and the output of 2.1 corresponds to the horizon test. Let's check this now. In fact, we will defend a slightly more specific claim: that the direct connection[5] of these heads to the input of the final layer norm corresponds to the count(test.

1a: Testing a simple hypothesis for heads 1.0, 2.0, 2.1

To start, we'll first test a simple hypothesis: that 1.0 and 2.0 just implement the simple Equal Count Test (notably, not the count(test) and 2.1 implements the horizon test, without

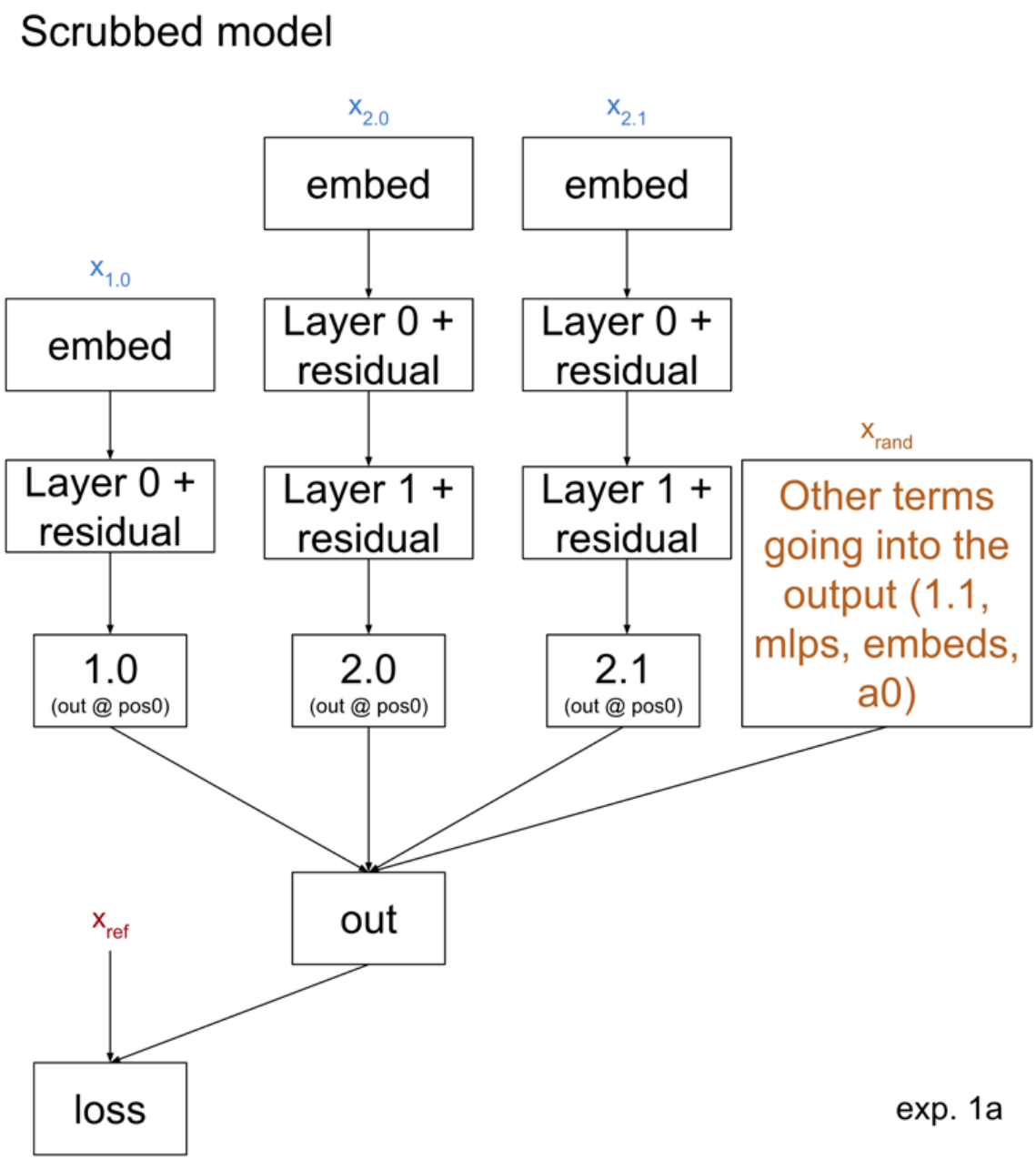
checking whether the sequence starts with an open parenthesis.

We can draw this claimed hypothesis in the following diagram (for the remainder of this doc we won't be drawing the inputs explicitly, to reduce clutter. Any node drawn as a leaf will have a single upstream input.):



The hypothesis proposed in Experiment 1---that head 1.0 and 2.0 implement the equal count test, while 2.1 implements the horizon test. In experiment 1a, we test the hypothesis via causal scrubbing. As before, we leave the performance metric (log loss) implicit.

How do we apply causal scrubbing to test this hypothesis? Let's walk through applying it for a single data point (a batch size of 1). We apply the causal scrubbing algorithm to this hypothesis and our model. This will choose 5 different input data points from the dataset described above, which we will use to run the tree-ified model on as shown below:



We first use the hypothesis from experiment 1a (that head 1.0 and 2.0 implement the equal count test and 2.1 implements the horizon test) to treeify our model (that is, split it into independent paths). We then scrub the model by replacing the outputs of head 1.0 and 2.0 with those of random samples that agree with the input on the count test, and 2.1 with those of random samples that agree with the input on the horizon test.

- **x_{ref}, or the reference input**. We compute the loss of the scrubbed model from the true label of x_{ref}. However, we will never run the scrubbed model on it; all inputs to the scrubbed model will be replaced with one of the other sampled inputs.
- Our hypothesis claims that, if we replace the output of 1.0 or of 2.0 with its output on some input x' that agrees with x_{ref} on the count test, then the output will agree with x_{ref} on the balanced test. Therefore we **sample random $x_{1.0}$ and $x_{2.0}$ which each**

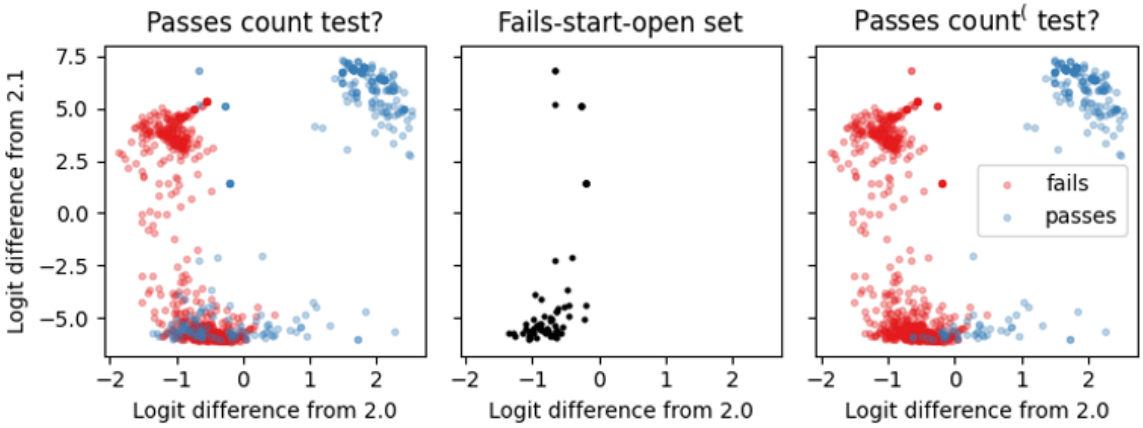
agree with x_{ref} on the count test. (Note that this means $x_{1.0}$ and $x_{2.0}$ agree with *each other* on this test as well, despite being separate inputs.)
- Similarly, **$x_{2.1}$ is sampled randomly conditional on agreeing with x_{ref} on the horizon test**.
- **x_{rand} is a random dataset example**. The subtree of the model which is rooted at the output and omits the branches included in the hypothesis–that is, the branches going directly to 1.0, 2.0, and 2.1–is run on this example.

We perform the above sample-and-run process many times to calculate the overall loss. We find that the scrubbed model recovers 88% of the original loss. The scrubbed model is very biased towards predicting unbalanced, with loss of 0.25 on unbalanced samples and 1.31 on balanced samples.

1b: Additional check performed by 1.0 and 2.0: initial parenthesis open

This, however, was still testing if 1.0 and 2.0 care about only the equal count test! As described above we believe it is more accurate to say that they check the count(test, testing that the first parenthesis is open as well as performing the Equal Count Test.

Consider the set of inputs that pass the equal count test but fail the count(test.[6] Let us call these the fails-start-open set. If we return to the attribution results from the informal hypothesis we can get intuition about the model's behavior on these inputs:



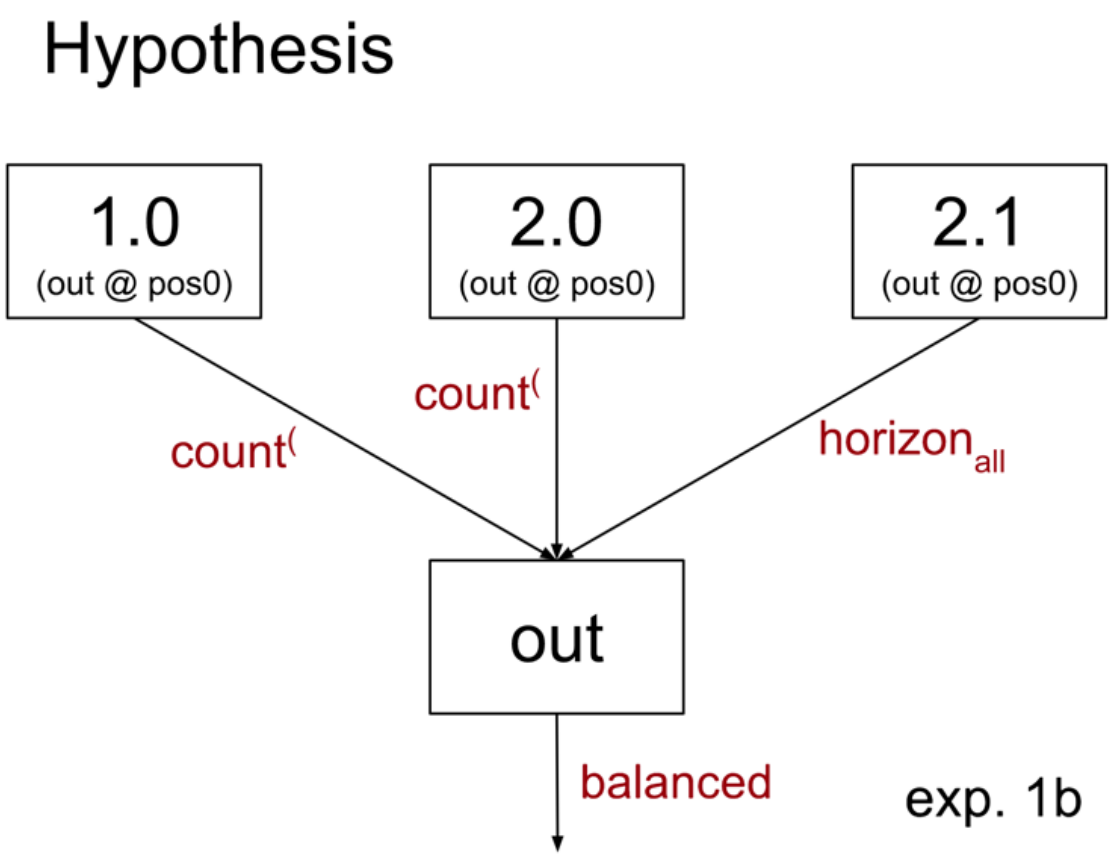
We plot parentheses sequences by the output of heads 2.0 and 2.1. We color the points by whether or not they pass a specific test. In the left most figure, we apply the simple equal count test while in the right most figure we color points by whether they pass the count^(test. In the center figure, we plot the points corresponding to sequences that start with ")" and thus always fail.

The left most figure suggests that head 2.0 is not implementing the count test, as it has a negative output on many points that pass the count test. As the center figure shows, these points correspond to points that start with ")". On the other hand, the count^(test more accurately predicts the logits of head 2.0. Taken together, these results suggest that head 2.0 is implementing the count^(test and not the count test.

The only difference from the left subfigure to the right subfigure is that the points in the fails-start-open set fail the more specific count(test. We see that indeed, the output of head 2.0 is

unbalanced on these inputs. Comparing the right and left diagrams we see the count(test more cleanly predicts the output of head 2.0. The results for head 1.0 are similar.

Here is an updated hypothesis claim:



Compared to experiment 1a, the intervention is different in two ways:

1. When the reference input is balanced, we may no longer sample $x_{1.0}$ or $x_{2.0}$ from the fails-start-open set. To the extent that our hypothesis is right and such inputs $x_{1.0}$ or $x_{2.0}$ cause 1.0 or 2.0 to output more "evidence of unbalance", this change will improve our loss on balanced sequences. Eyeballing the plots above, we do expect this to happen.
2. When the reference input is unbalanced, we may now sample $x_{1.0}$ and $x_{2.0}$ from the fails-start-open set. To the extent that our hypothesis is wrong and such inputs cause 1.0 or 2.0, respectively, to output *less* "evidence of unbalance", this change will harm our loss on unbalanced sequences. Eyeballing the plots above, it is somewhat hard to tell whether we should expect this: these points do have some evidence of unbalance, but it is unclear how the magnitude compares to that of the fail-count set.

The scrubbed model recovers 93% of the loss. In particular the loss on balanced goes way down (1.31 -> 0.65) while the loss on unbalanced is slightly lower (0.25 -> 0.18). Thus this experiment supports our previous interpretability result that 1.0 and 2.0 detect whether the first parenthesis is open.

Comparing experiments 1a and 1b makes it clear the count(test is an improvement. However, it is worth noticing that if one had only the results of experiment 1a, it would not be clear that such an improvement needed to be made.[7] In general, causal scrubbing is not

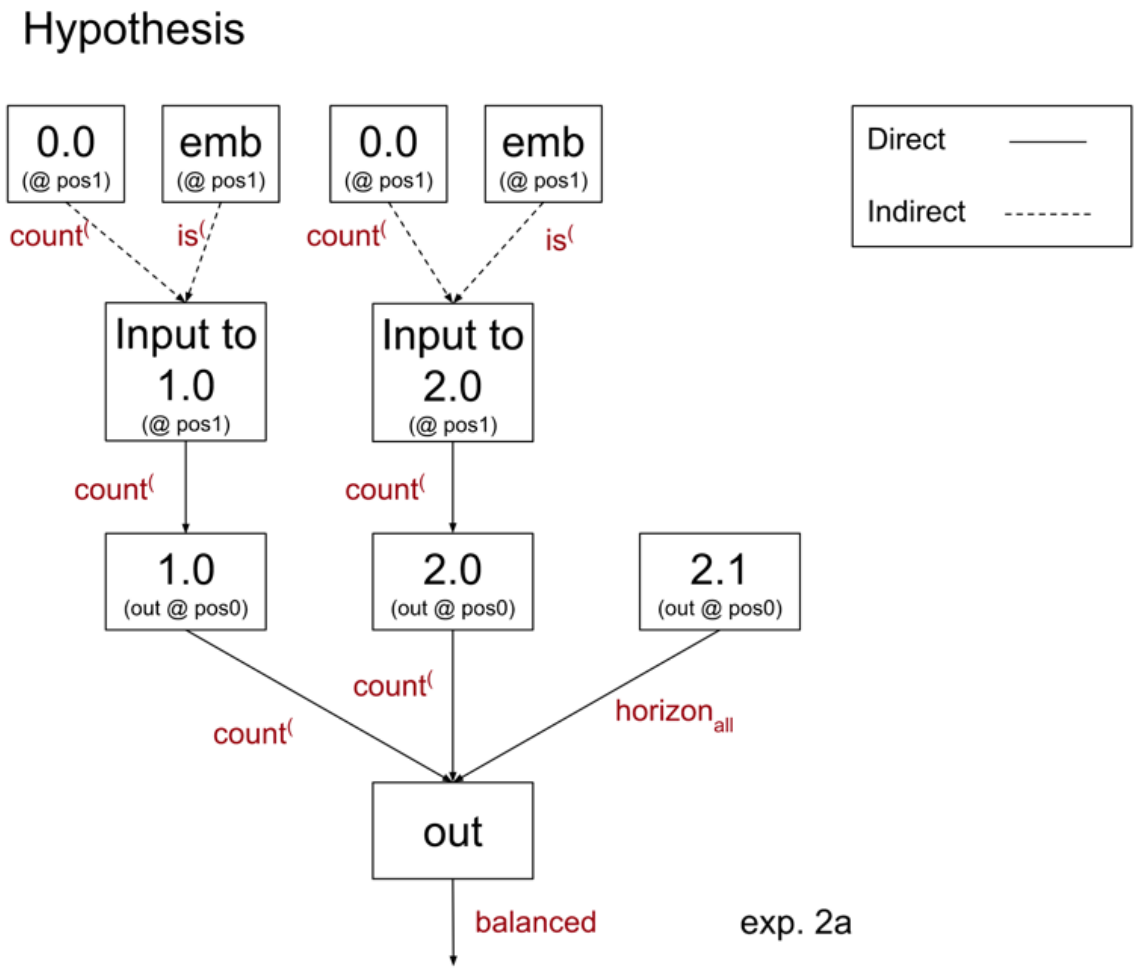
legibly punishing when the feature you claimed correspondance with is highly correlated with the ‘true feature’ that the component of the model is in fact picking up. We expect that all our claims, while highly correlated with the truth, will miss some nuance in the exact boundaries represented by the model.

Experiment 2: More specific explanation for 1.0’s and 2.0’s input

To make the above hypothesis more specific, we’ll explain how 1.0 and 2.0 compute the count(test: in particular, they use the output of 0.0 at position 1. To test this, we update the hypothesis from 1a to say that 2.0 and 1.0 only depend on whether the first parenthesis is open and whether 0.0 is run on an input that passes the count(test. The other inputs to the subtrees rooted at 1.0 and 2.0 don’t matter. We aren’t stating *how* the output of 0.0 and the embedding reach those later heads; we’re considering the indirect effect, i.e. via all possible paths, rather than just the direct effect that passes through no other nodes in the model.

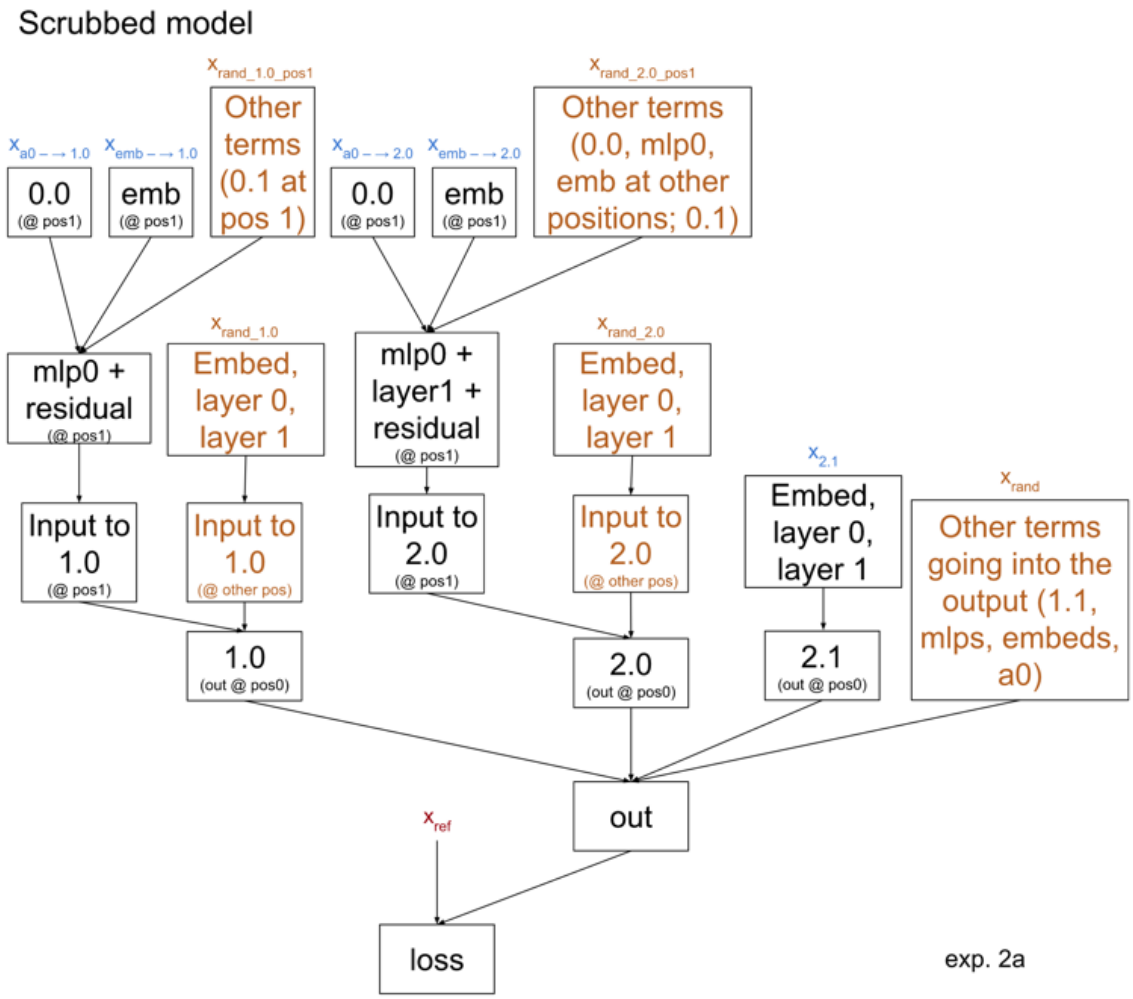
2a: Dependency on 0.0

Our claimed hypothesis is shown below. Recall that we do not show unimportant nodes, we annotate the nodes of I with the nodes of G that they correspond to, and we annotate the edges with the feature that that node of I computes:[8]



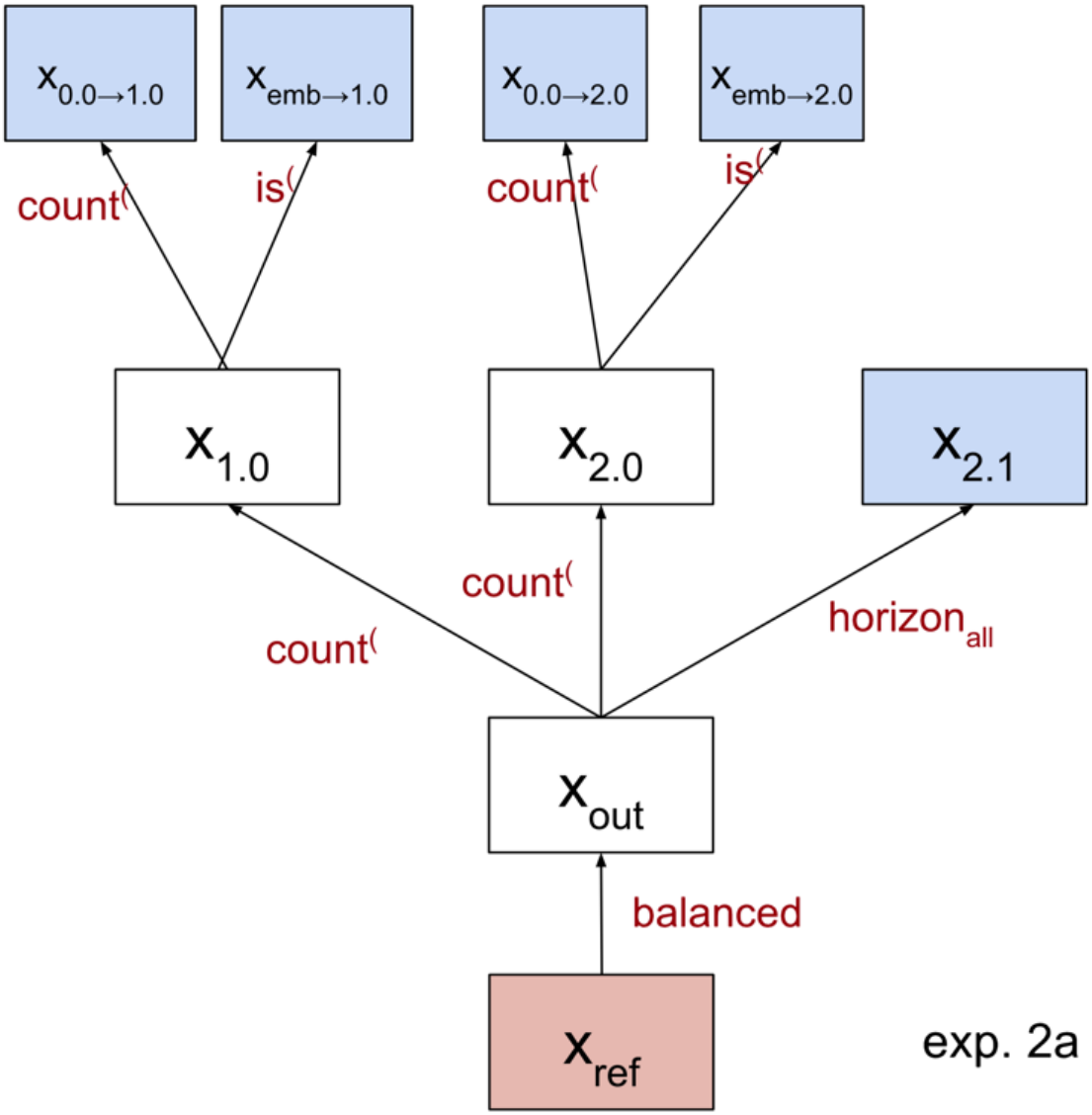
A more specific hypothesis for the parens balancer model. Namely, we hypothesize that head 1.0 and 2.0 implement the count(test by reading off head 0.0's output at sequence position 1.

This hypothesis will result in the following treeified model:



We apply causal scrubbing to our refined hypothesis (that head 1.0 and 2.0 implement the count^(test by reading off head 0.0's output at sequence 1) to get this scrubbed model.

How do we determine the 5 input data points annotated in blue? Following the causal scrubbing algorithm, we first fix x_{ref} at the output. We then recursively move through the hypothesis diagram, sampling a dataset example for *every* node such that it agrees with the downstream nodes on the labeled features. The non-leaf node data points can then be discarded, as they are not used as inputs in the scrubbed model. This is depicted below:



In particular, we will first choose a dataset sample x_{ref} whose label we will use to evaluate the loss of the scrubbed model. Then we will select the input datasets as follows:

- x_{out} agrees with x_{ref} on the balanced test.
- $x_{2.1}$ agrees with x_{ref} on the never beneath horizon test (as before)
- Both $x_{1.0}$ and $x_{2.0}$ agree with x_{ref} on the count(test.
- $x_{0.0\to1.0}$ agrees with $x_{1.0}$ on the count(test, and similarly for 2.0.
- $x_{emb\to1.0}$ agrees with $x_{1.0}$ on whether the sequence starts with (, and similarly for 2.0.

Note that we do not require any other agreement between inputs! For example, $x_{emb\to1.0}$ could be an input that fails the count test.

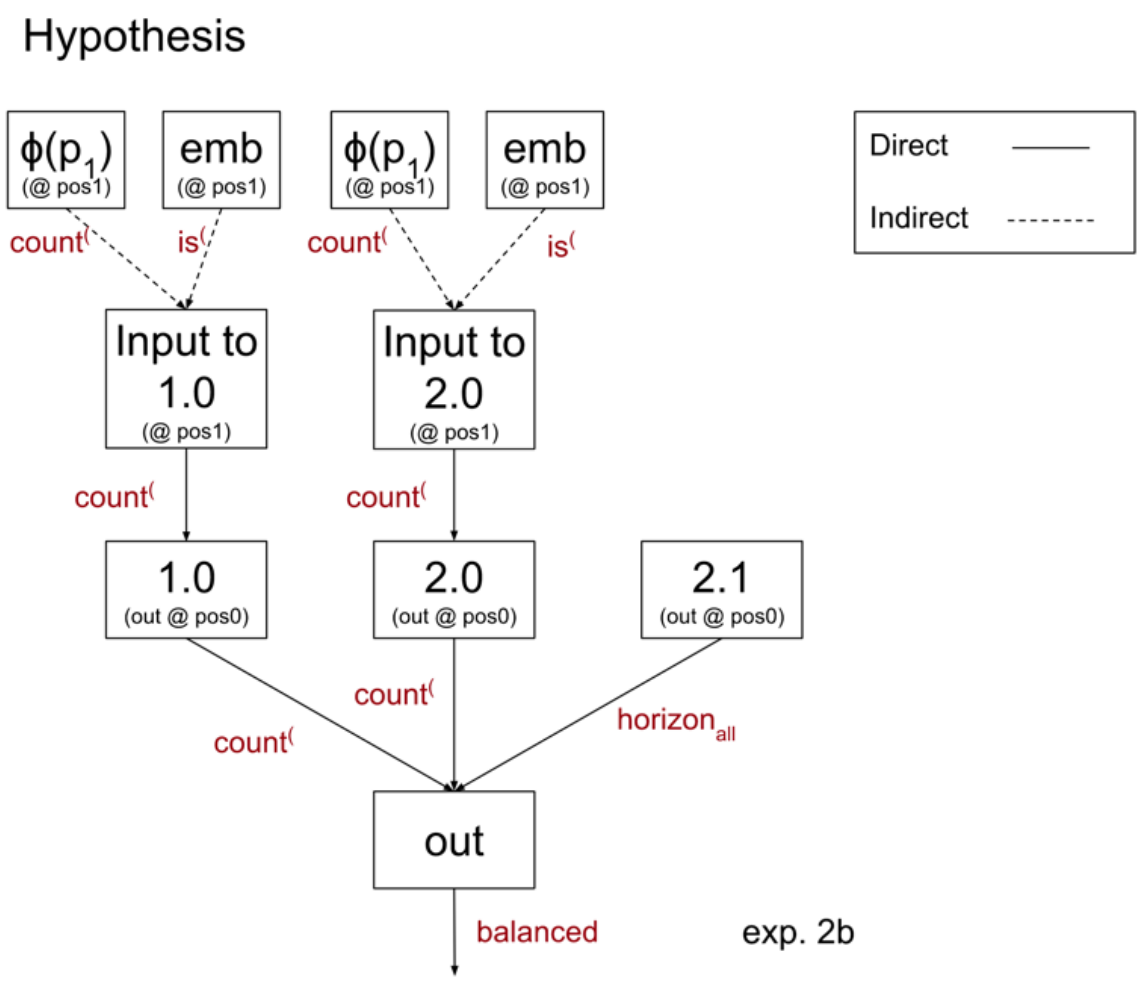
The 5 inputs in orange are claimed to be unimportant by our hypothesis. This means we will sample them randomly. We do, however, use the *same* random value for all unimportant inputs to a particular node in our model. For instance, there are many 'unimportant' inputs to the final layer norm: all three mlps, attention layer 0, and head 1.1. All of these are sampled

together. Meanwhile, we sample these nodes *separately* from unimportant inputs to other nodes (e.g. the non-position 1 inputs to head 2.0); see the appendix for some discussion of this.

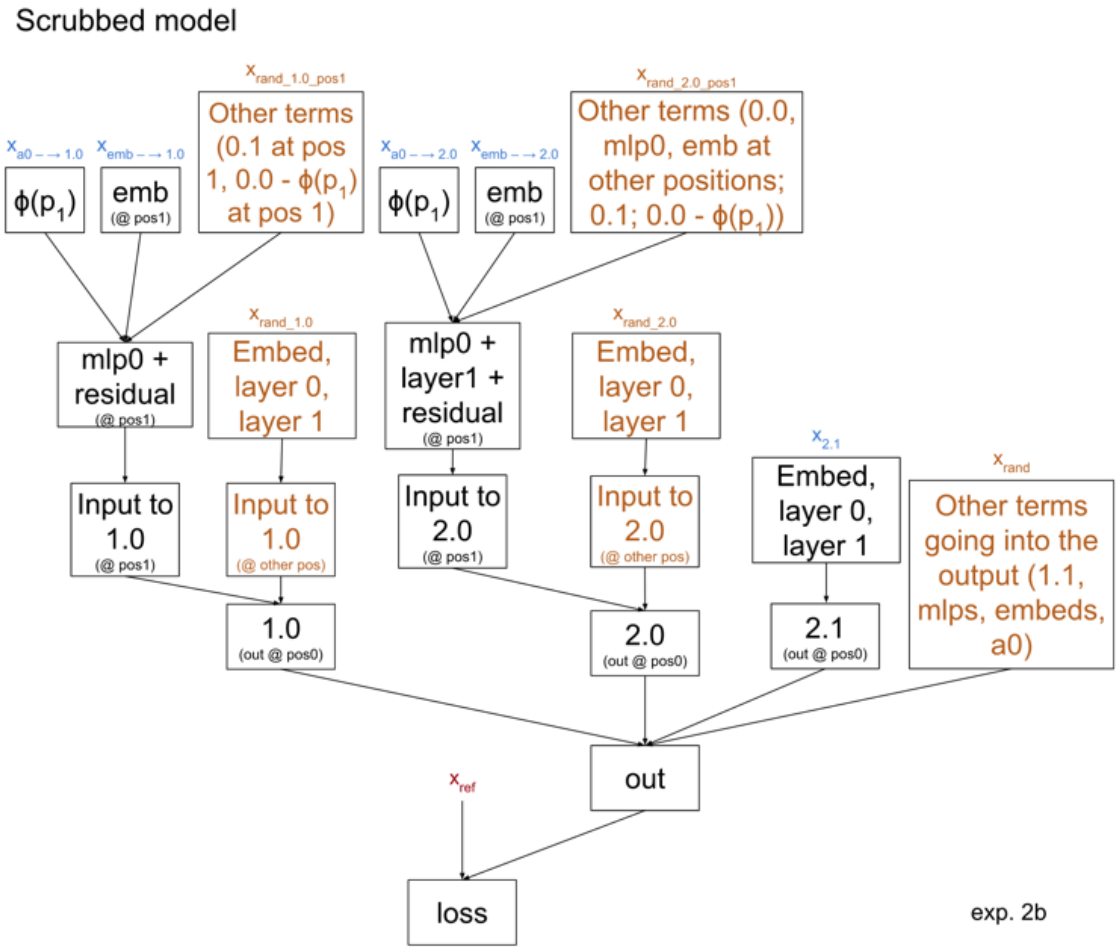
The scrubbed model recovers 88% of the loss. Compared to experiment 1b, the loss recovered is significantly lower: a sign that we lost some explanatory power with this more specific hypothesis. By asserting these more specific claims, however, we still recover a large portion of the original loss. Overall, we think this result provides evidence that our hypothesis is a reasonable approximation for the 2.0 circuit. (To rule out the possibility that 0.0 is not important at all in these paths, we also ran an experiment replacing its output with that on a random input; this recovered 84% of the loss which is significantly less).

2b: 0.0's output encodes p in a specific direction

In fact, we believe that the proportion of open parentheses is encoded linearly in a particular direction. For more details and a precise definition of how we think this is done, see the appendix. The takeaway, however, is that we have a function ϕ which maps a value of p to a predicted activation of 0.0. We can thus rewrite the output of 0.0 in our model as the sum of two terms: $\phi(p)$ and the residual (the error of this estimate). We then claim that the $\phi(p)$ term is the important one. In essence, this allows swapping around the residuals between any two inputs, while $\phi(p)$ can only be swapped between inputs that agree on the count(test. As a hypothesis diagram, this is:



Which leads to the following treeified model (again, with unimportant nodes in orange):



This results in a loss of 0.56, with accuracy 87%. This is basically unchanged from experiment 2a, giving evidence that we were correct in how p values are translated into the output of 0.0. Importantly, however, if we were somewhat wrong about *which* p values head 0.0 outputs on each input, this would have already hurt us in experiment 2a. Thus this result shouldn't increase our confidence on that account.

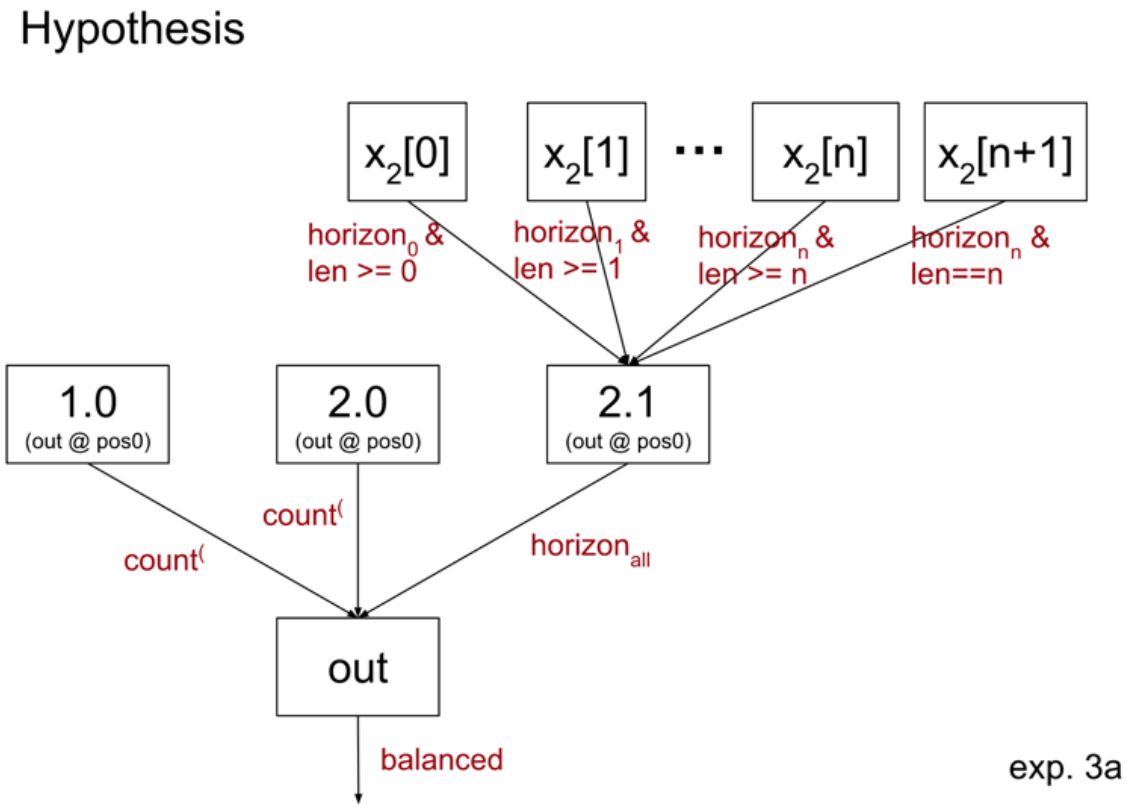
Experiment 3: How does 2.1 compute the horizon condition?

For this experiment, we are not including the breakdown of 2.0 from experiment 2. We will add these back in for experiment 4, but it is simpler to consider them separately for now.

3a: Breaking up the input by sequence position

From previous interpretability work, we think that 0.0 computes the proportion of open parentheses in the suffix-substring starting at each query position. Then, mlp0 and mlp1 check the not-beneath-horizon test at that particular position. This means that 2.1 needs to ensure the check passes at every position (in practice, the attention pattern will focus on failed positions, which cause the head to output in an unbalanced direction).

We test this by sampling the input to head 2.1 at every sequence position separately (with some constraints, discussed below). This corresponds to the following hypothesis:



where $x_2[i]$ denotes the input to attention 2 at position i, and n is the number of parentheses in the sequence. In particular, we fix n per example when we choose the dataset $x_{2.1}$. We additionally decide that $x_2[i]$ must be at least i-parentheses long for all i <= n to avoid OOD edge cases that we didn't wish to make claims about e.g. samples including multiple `[END]` tokens (possibly a weaker constraint would be sufficient, but we have not experimented with that).

One other subtlety is what to do with the last sequence position, where the input is a special `[END]` token. We discovered that this position of the input to 2.1 carries some information about the last parenthesis.[9] We allow interchanges between different `[END]` positions as long as they agree on the last parenthesis. This is equivalent to requiring agreement on both the horizon_n test and that the sequence is exactly len_n.

The causal scrubbing algorithm is especially strict when testing this hypothesis. Since 2.1 checks for *any* failure, a failure at a single input sequence position should be enough to cause it to output unbalance. In fact our horizon_i condition is not quite true to what the model is doing, i.e. 2.1 is able to detect unbalanced sequences based on input at position i even if the horizon_i test passes. Even if the horizon_i condition is most of what is going on, we are likely to sample at least one of these alternative failure detections because we sample up to 40 independent inputs, leading head 2.1 to output unbalance most of the time!

The overall loss recovered from doing this scrubbing is 77%. The model is again highly skewed towards predicting unbalanced, with a loss of 3.61 on balanced labels.

3b: Refining our notion of the open-proportion

We can improve this performance somewhat by shifting our notion of horizon_i to one closer to what the model computes. In particular our current notion assumes the attention pattern of 0.0 is perfectly upper triangular (each query position pays attention evenly across all later key positions). Instead, it is somewhat more accurate to describe it as 'quasi upper triangular': it pays *more* attention to the upper triangular positions, but not exclusively. This relaxed assumption gives rise to a new "adjusted p" value that we can substitute for p in our interpretation; see the appendix. It turns out the new I still correctly computes if an input is balanced or not.

Using this new hypothesis improves our loss recovery to 84%, a notable increase from experiment 3a. Breaking up the loss by balanced and unbalanced reference sequences, we see that the loss decreased specifically on the balanced ones.

3c and 3d: Making the hypothesis more specific

We additionally ran experiments where we split up the input $x_2[i]$ into terms and specified how it was computed by the MLPs reading from a0 (similar to experiment 2). Counterintuitively, this *decreases* the loss.

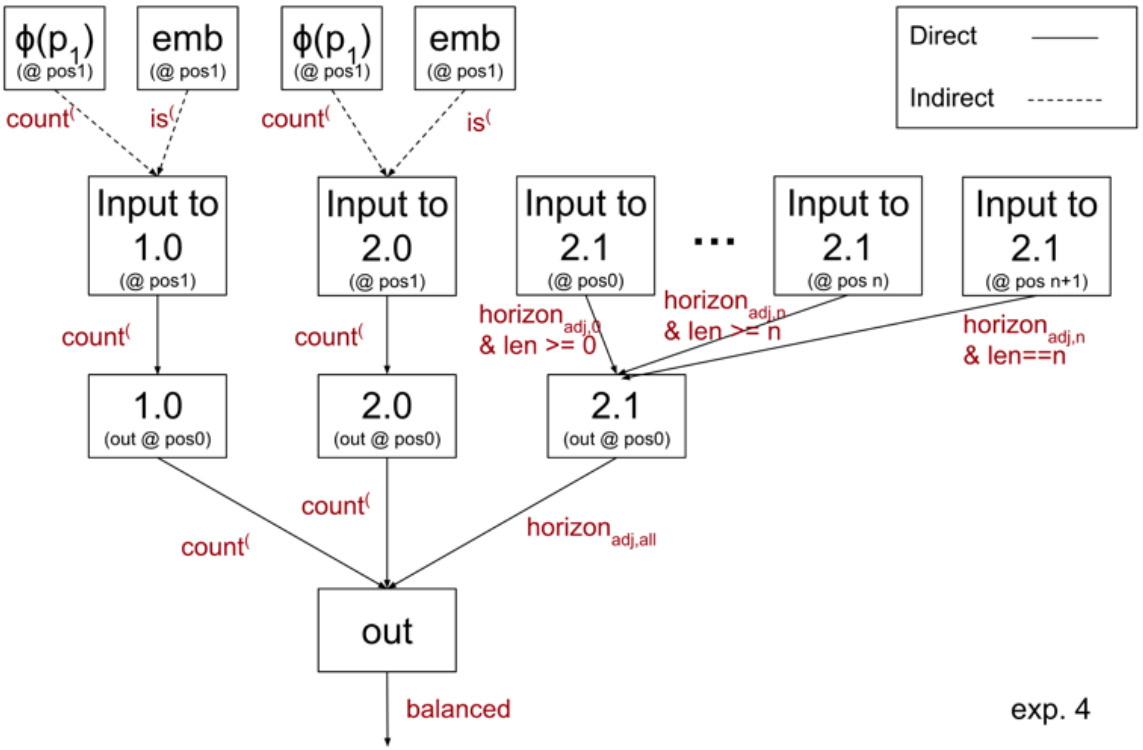
In general, causal scrubbing samples inputs separately when permitted by the hypothesis. This, however, is a case where sampling *together* is worse for the scrubbed performance.

More detail on these experiments, and their implications, can be found in the appendix.

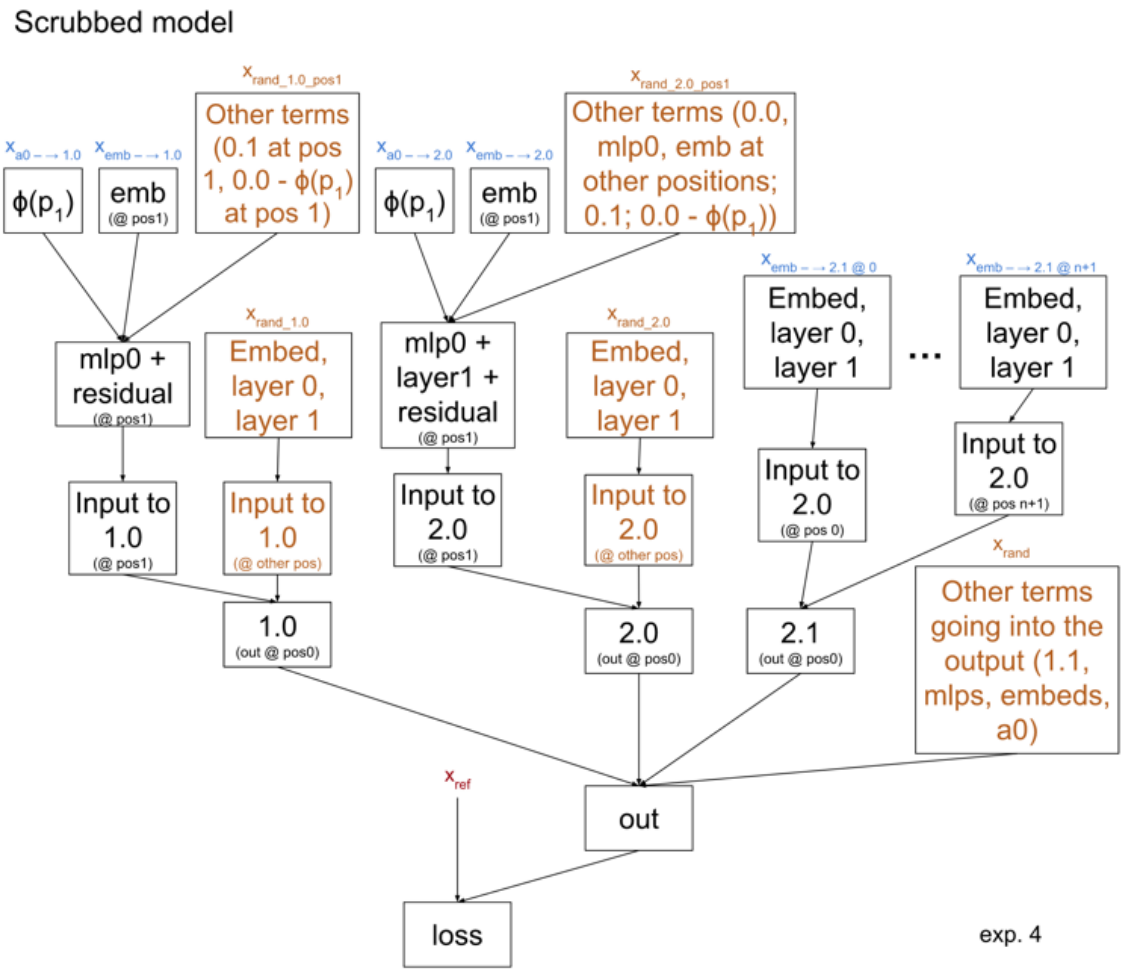
Experiment 4: putting it all together

We can combine our hypotheses about what 2.0 is doing (experiment 2b) and what 2.1 is doing (experiment 3b) into a single hypothesis:

Hypothesis



exp. 4



This results in 72% loss recovered. We note that the *loss* is roughly additive: the loss of the scrubbed model in this experiment is roughly the sum of the losses of the two previous experiments.

Future Work

There are still many ways our hypothesis could be improved. One way would be to make it more comprehensive, by understanding and incorporating additional paths through the model. For example, we have some initial evidence that head 1.1 can recognize some horizon failures and copy this information to the residual stream at position 1, causing head 2.0 to output the sequence is unbalanced. This path is claimed to be unimportant in Experiment 2, which likely causes some of the loss increase (and corresponding decrease in % loss recovered).

Additionally, the hypothesis could be made more specific. For instance in the appendix we make more specific claims about exactly how head 0.0 computes p; these claims would be possible to test with causal scrubbing, although we have not done so. Similarly, it would be possible to test very specific claims about how the count or horizon_i test is computed from head 0.0, even at the level of which neurons are involved. In particular, the current hypothesized explanation for 2.1's input is especially vague; replicating the techniques from experiment 2 on these inputs would be a clear improvement.

Another direction we could expand on this work would be to more greatly prioritize *accuracy* of our hypothesis, even if it comes at the cost of interpretability. In this project we have kept to a more abstract and interpretable understanding of the model's computation. In particular for head 0.0 we have approximated its attention pattern, assuming it is (mostly) upper triangular. We could also imagine moving further in the direction we did with p_{adj} and estimating the attention probabilities 0.0 will have position by position. This would more accurately match the (imperfect) heuristics the model depends on, which could be useful for predicting adversarial examples for the model. For an example of incorporating heuristics into a causal scrubbing hypothesis, see our results on induction in a language model.

Conclusion

Overall, we were able to use causal scrubbing to get some evidence validating our original interpretability hypothesis, and recover the majority of the loss. We were also able to demonstrate that some very specific scrubs are feasible in practice, for instance rewriting the output of 0.0 at position 1 as the sum of $\phi(p)$ and a residual.

Using causal scrubbing led us to a better understanding of the model. Improving our score required refinements like using the adjusted open proportion or including that the end-token sequence position can carry evidence of unbalance to 2.1 in our hypothesis.

This work also highlighted some of the challenges of applying causal scrubbing. One recurring challenge was that scores are not obviously good or bad, only better or worse relative to others. For example, in our dataset there are many features that could be used to distinguish balanced and unbalanced sequences; this correlation made it hard to notice when we specified a subtly wrong feature of our dataset, as discussed when comparing experiments 1a and 1b, since the score was not obviously bad. This is not fundamentally a problem–we did in fact capture a lot of what our model was doing, and our score was reflective of that–but we found these small imprecisions in our understanding added up as we made our hypothesis more specific.

We also saw how, in some cases, our intuitions about how well we understood the model did not correspond to the loss recovered by our scrubbed model. Sometimes the scrubbed model's loss was especially sensitive to certain parts (for instance, unbalanced evidence in the input to head 2.1 at a single sequence position) which can be punishing if the hypothesis isn't perfectly accurate. Other times we would incorporate what we expected to be a noticeable improvement and find it made little difference to the overall loss.

Conversely, for experiments 3c and 3d (discussed in the appendix) we saw the scrubbed model's loss decrease for what we ultimately believe are unjustified reasons, highlighting the need for something like adversarial validation in order to have confidence that a hypothesis is actually good.

In general, however, we think that these results provide some evidence that the causal scrubbing framework can be used to validate interpretability results produced by more ad-hoc methods. While formalizing the informal claims into testable hypotheses takes some work, the causal framework is remarkably expressive.

Additionally, even if the causal scrubbing method only validates claims, instead of producing them, we are excited about the role it will play in future efforts to explain model behaviors. Having a flexible but consistent language to express claims about what a model is doing has many advantages for easily communicating and checking many different variations of a hypothesis. We expect these advantages to only increase as we build better tools for easily expressing and iterating on hypotheses.

Appendix

Data set details

This model was trained with binary cross-entropy loss on a class-balanced dataset of 100k sequences of open and close parens, with labels indicating whether the sequence was balanced. The tokenizer has 5 tokens: `( ) [BEGIN] [END] [PAD]`. The token at position 0 is always `[BEGIN]`, followed by up to 40 open or close paren tokens, then `[END]`, then padding until length 42.

The original dataset was a mixture of several different datasets with binary cross entropy loss:

1. Most of the training data (~92%) was randomly-generated sequences with an even number of parentheses. Balanced sequences were upsampled to be about 26% of the dataset.
2. Special case datasets (the empty input, odd length inputs)
3. Tricky sequences, which were adversarial examples for other models.

For the experiments in this writeup we will only use the first dataset of randomly generated inputs. We are attempting to explain the behavior of "how does this model get low cross-entropy loss on this randomly-generated dataset." This may require a subtly different explanation than how it predicts more difficult examples.

Code

We plan to release our code and will link it here when available. Note that the computation depends on our in-house tensor-computation library, expect it to be time consuming to understand the details of what is being computed. Feel free to get in contact if it is important for you to understand such things.

Sampling unimportant inputs

In our previous post, we discussed reasons for sampling unimportant inputs to a node in our model (specifically, in $c(I) \subset G$) *separately* from unimportant inputs to other nodes in our correspondence.

In this work, this was very important for reasoning about what correlations exist between different inputs and interpreting the results of our experiments. Consider the hypotheses 3c and 3d. If we claim that the inputs to 2.1 at each position carry information about the horizon_i test, then each a1_i will be sampled separately. If we claimed instead that only the mlps and a0 had that job, and a1 was unimportant, we would still like each a1_i to be sampled separately! That way the two claims differ *only* in whether a1 is sampled conditional on the horizon_i test, and not in whether the a1_i are drawn from the same input.

In those experiments we discuss how the correlation between inputs hurt the loss of our scrubbed model. In fact, experimentally we found that if we ran 3d but sampled a1 across positions together, it hurt our scrubbed model's loss. If we had run this experiment alone, without running 3d, the effects of "sampling a1[i] separately from the other terms at position i" and "sampling the a1[i] all together" would be confounded.

So, sampling unimportant inputs separately is especially important for comparing the swaps induced by hypotheses 3c and 3d cleanly. The choice makes minimal difference elsewhere.

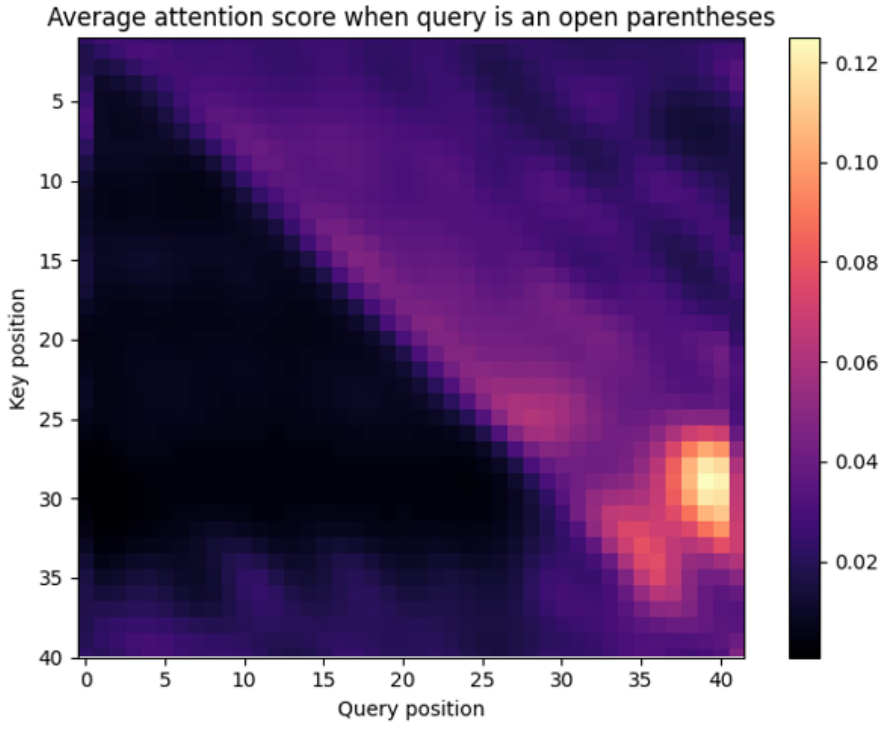
Analysis of Head 0.0

The attention pattern of layer zero heads is a relatively simple function of the input. Recall that for every (query, key) pair of positions we compute an attention score. We then divide by a constant and take the query-axis softmax so that the attention paid by every query position sums to one. For layer 0 heads, each attention score is simply a function of four inputs: the query token, the query position, the key token, and the key position:

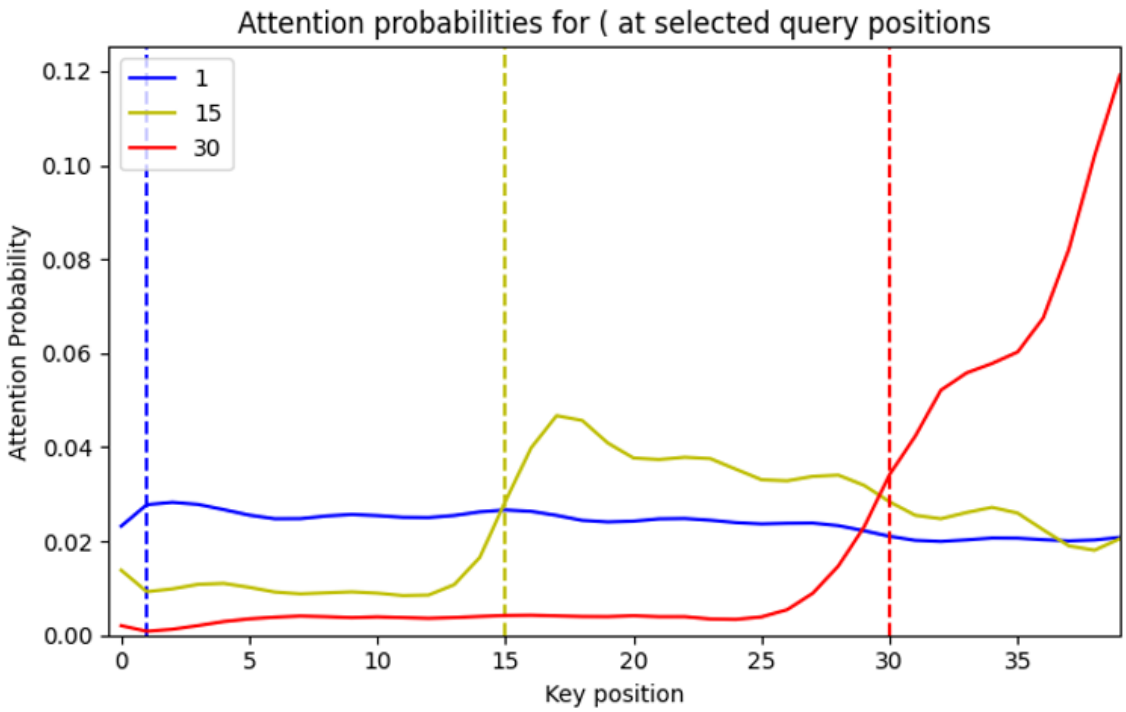
$$s(q_{\text{tok}}, q_{\text{pos}}, k_{\text{tok}}, k_{\text{pos}}) = (W^Q_{0.0}\text{LN}_0(\text{emb}_{q_{\text{tok}},q_{\text{pos}}}) + \beta^Q_{0.0}) \cdot (W^K_{0.0}\text{LN}_0(\text{emb}_{k_{\text{tok}},k_{\text{pos}}}) + \beta^K_{0.0})$$

One pattern that is noticeable is that if the query is an open parentheses, the attention score does not change based on if the key token is an open or close parentheses. That is, $s(\text{open}, q_{\text{pos}}, \text{open}, k_{\text{pos}}) \approx s(\text{open}, q_{\text{pos}}, \text{close}, k_{\text{pos}})$ for all possible query and key positions.

This means that the attention pattern at an open parentheses query will only depend on the query position and the length of the entire sequence. The expected attention pattern (after softmax) for sequences of length 40 is displayed below:



And focusing on three representative query positions (each one a row in the above plot):



Some things to notice:

- To a first approximation, the attention is roughly upper triangular.
- The attention before the query position is non-zero, but mostly flat. This will motivate our definition of p_{adj} for experiment 3b.
- There are various imperfections. We expect these are some of the reasons our model has non-perfect performance.

As a simplifying assumption, let us assume that the attention pattern *is* perfectly upper triangular. That is every query position pays attention to itself and all later positions in the sequence. What then would the head output?

One way to compute the output of an attention head is to first multiply each input position by the V and O matrices, and then take a weighted average of these with weights given by the attention probabilities. It is thus useful to consider the values

$$h_{k_{tok},i} = W^{O}_{0.0}(W^{V}_{0.0}LN_0(emb_{k_{tok},i}) + \beta^{V})$$

before this weighted average.

It turns out that $h_{k_{tok},i}$ depends strongly on if k_{tok} is an open or close parentheses, but doesn't depend on the position i. That is we can define $\bar{h}_{open}$ and $\bar{h}_{close}$ to be the mean across positions. All $h_{open,i}$ point in the direction of $\bar{h}_{open}$ (minimum cosine similarity is 0.994), and all $h_{close,i}$ point in the direction of $\bar{h}_{close}$ (minimum cosine similarity is 0.995). $\bar{h}_{open}$ and $\bar{h}_{close}$, however, point in opposite directions (cosine similarity -0.995).

We can combine what we have learned about the attention and the effect of the O and V matrices to give a reasonable understanding of the behavior of this head. Let us assume the attention pattern is perfectly upper triangular. Then at query position i, p_i of the attention will be on open parentheses positions, and (1-p_i) of the attention will be on close parentheses positions.[10] Then the output of the head will be well approximated by $p_i\bar{h}_{open} + (1 - p_i)\bar{h}_{close}$

Since these terms are in nearly-opposing directions, we can well approximate the activation in a rank-one subspace:

$$\phi(p_i) = (2p_i - 1)(\bar{h}_{open} - \bar{h}_{close})/2$$

This shows how 0.0 computes p_i at open parenthesis positions. We also directly test this ϕ function in experiment 2b.

p_{adj}: A more accurate replacement for p

In the previous appendix section we assumed the attention pattern of 0.0 is perfectly upper triangular. We did note that 0.0 pays non-zero attention to positions before the query.

Defining p_{adj}

Fix some query position q in an input of length n. We can split the string into a prefix and a suffix, where the prefix is positions [1, q-1] and the suffix is positions [q, n]. If 0.0 had a perfectly upper triangular attention pattern, it would pay 1/len(suffix) attention to every key position in the suffix.

Instead, however, let us assume that it pays $b_{q,n}$ attention to the prefix, leaving only (1-$b_{q,n}$) attention for the suffix. Then it pays 1/len(prefix) attention to every position in the prefix, and (1-$b_{q,n}$)/len(suffix) attention to every position in the suffix.

We calculate every $b_{q,n}$ based on analysis of the attention pattern. Note these are independent of the sequence. Two important facts are true about these values:

1. $b_{1,n} = 0$. That is, at position 1 no attention is paid to the prefix, since no prefix exists.
2. $b_{q,n} < (q-1)/n = \text{len(prefix)}/n$ This implies that at every position, for every sequence length, more attention is paid to a given position in the suffix than in the prefix.

We then define $p_{adj,\,q}$ based on this hypothesized attention. If p_{prefix} and p_{suffix} are the proportion of open parentheses in the respective substrings, then

$$p_{adj,q} = b_{q,n}p_{prefix} + (1 - b_{q,n})p_{suffix}$$

The adjusted I

The count test is unchanged, since fact 1 above implies $p_{adj,1} = p_1$. The never-beneath-horizon test is altered: we now test $\text{horizon}_{adj,i}$ which is defined to be true if $p_{adj,i} \leq 0.5$. While this doesn't agree on all sequences, we will show it does agree for sequences that pass the count test. This is sufficient to show that our new I always computes if a given input is balanced (since the value of the horizon test is unimportant if the count test fails).

Thus, to complete the proof, we will fix some input passes the count test and a query position q. We will show that the adjusted horizon test at q passes exactly if the normal horizon test at q passes.

We can express both p_1 and $p_{adj,q}$ as weighted averages of p_{prefix} and p_{suffix}. In particular,

$$p_1 = \frac{q_n=1}{} p_{prefix} + (1 - \frac{q_n=1}{}) p_{suffix}$$

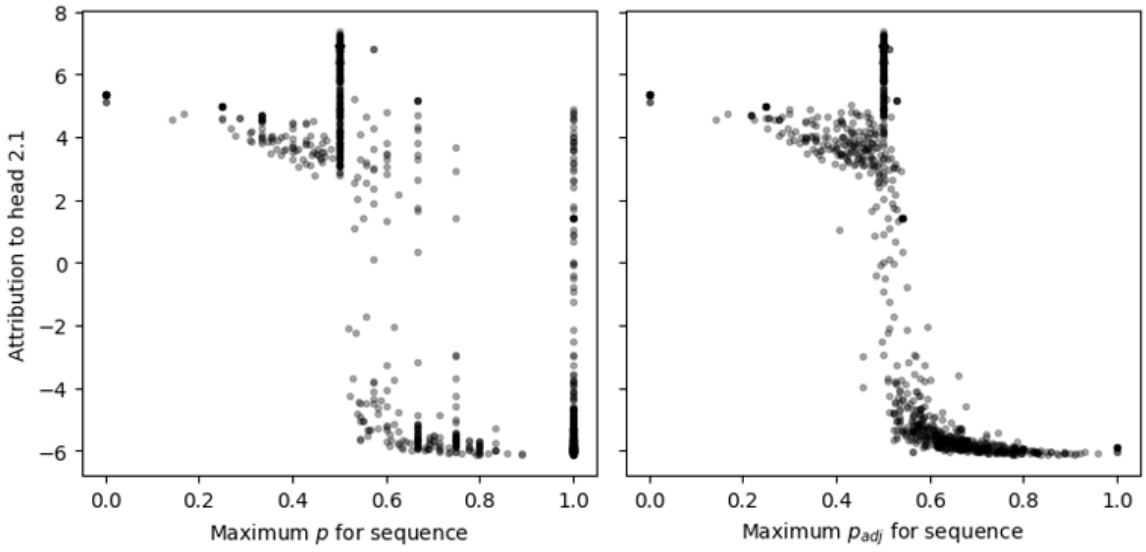
$$p_{adj,q} = b_{q,n} p_{prefix} + (1 - b_{q,n}) p_{suffix}$$

However, $b_{q\,n} < (q-1)/n$. Thus, $p_{adj,q} > p1$ exactly when $p_{suffix} > p_{prefix}$. Since the input passes the count test, $p^1=0.5$ which implies only one of p_{suffix} and p_{prefix} can be greater than 0.5. Thus, a horizon failure at q $\Leftrightarrow$ $p_{suffix} > 0.5 \Leftrightarrow p_{suffix} > p_{prefix} \Leftrightarrow p_{adj,\,q} \Leftrightarrow$ an adjusted horizon failure at q.

This shows the horizon tests agree at every position of any input that passes the count test. This ensures they agree on if any input is balanced, and our new causal graph is still perfectly accurate.

Attribution Results

For some evidence that the adjusted proportion more closely matches what 2.1 uses, we can return to [our measure of the logit difference to 2.1]. We might hope that the maximum value of p_i across the sequence has a clear correspondence with the output of 2.1. However, it turns out there are many sequences that end in an open parentheses (and thus $p_n=1$) but 2.1 does not clearly output an unbalanced signal, as can be seen in the left subplot below:



In practice, these are often sequences with many more close parentheses than open parentheses. Thus, even at the last position 0.0 attention will mostly be spread among positions with close parentheses. While this means 2.1 may not pick up on the failure, 2.0 will be able to detect these sequences as unbalanced.

This type of dynamic is captured in our definition for p_{adj}. We can see that the maximum adjusted proportion has a much clearer relationship with the attribution to head 2.1.

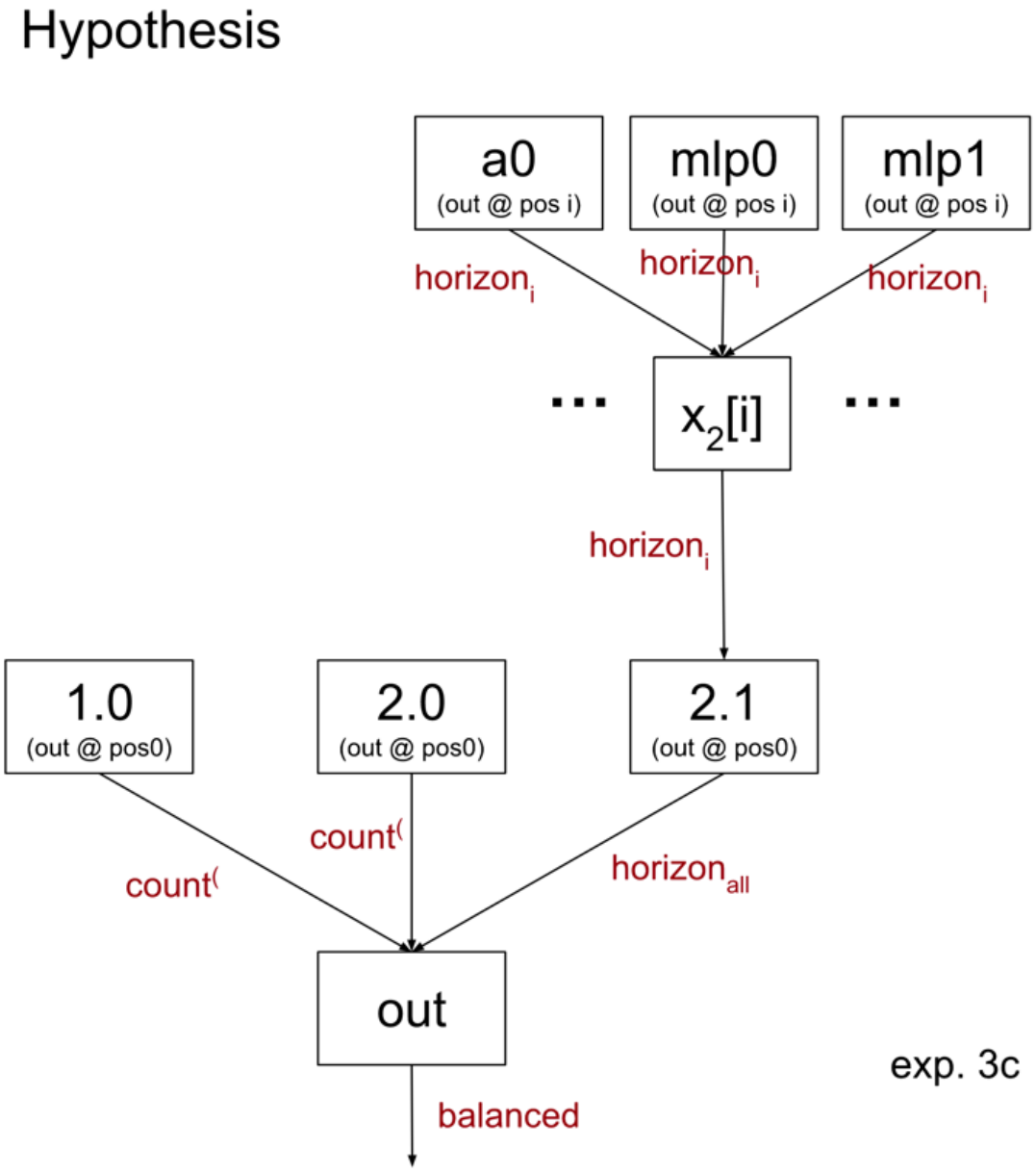
Effect of p_{adj} on scrubbed model loss

The plot above does not explain why our scrubbed model performs better when using p_{adj}; the lower loss comes from samples that are not on the maximum p or p_{adj} for the sequence. In particular the attribution plot has clearer separation of classes because we remove false-negatives of the original horizon test at the sequence level ($horizon_{all}$ fails but 2.1 does not say the input is unbalanced; these are removed because $horizon_{adj}$ passes). The main reason the scrubbed loss improves, however, is because we remove false-positives at the position level ($horizon_i$ passes but 2.1 treats the input a failure; these are removed because $horizon_{adj,i}$ fails).

Examples where $horizon_i$ passes but $horizon_{adj,i}$ fails are ones where there is a horizon failure somewhere in the prefix. Thus, there aren't sequence level false positives of the horizon test (when compared to the adjusted horizon test). In practice the shortcomings of the normal horizon test seem to not be a problem for experiments 1 and 2.[11] It is notably worse for experiment 3, however, where sampling a single $x_2[i]$ that has unbalanced-evidence is enough to cause the model to flip from a confidently balanced to confidently unbalanced prediction.

Breaking up Experiment 3 by term

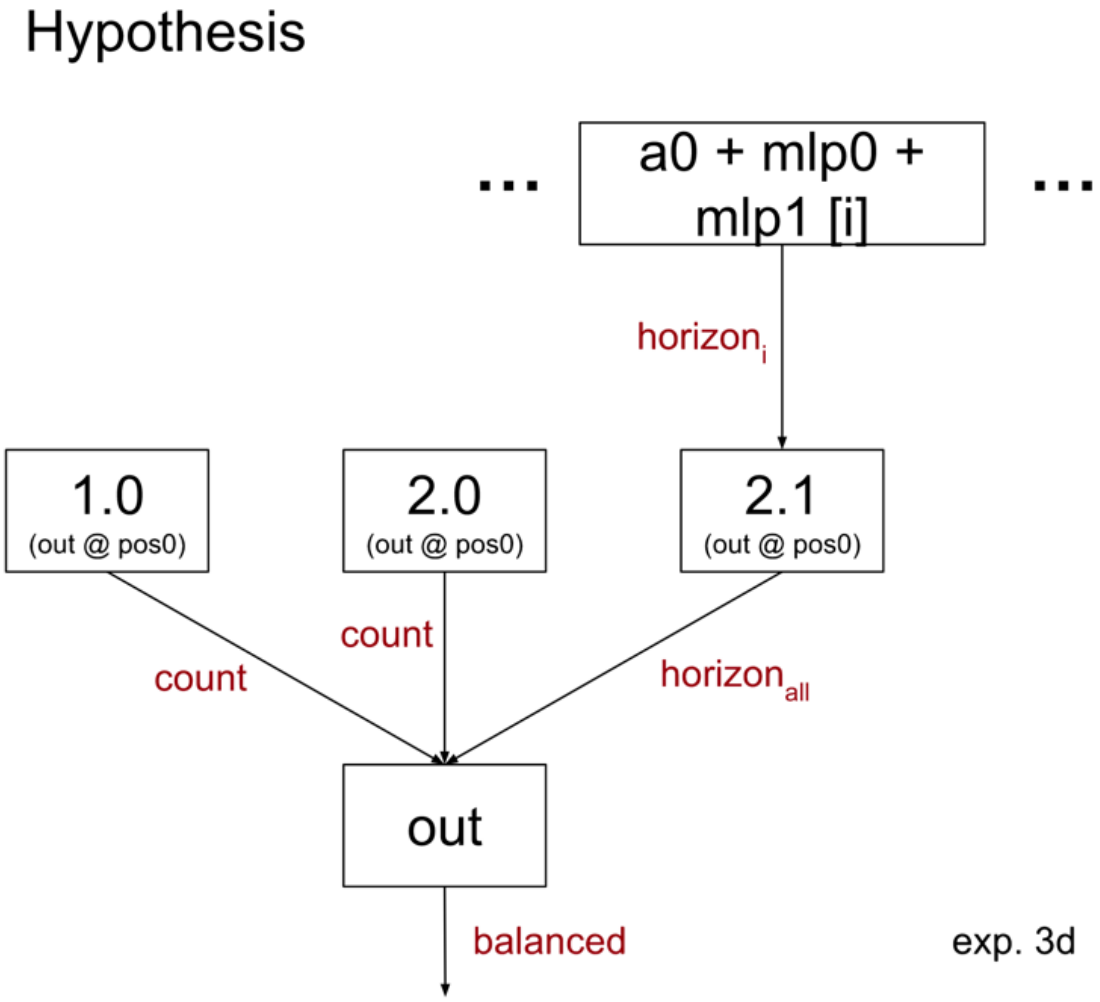
In order to make our hypothesis 3a more specific we can claim that the only relevant parts of $x_2[i]$ are the terms from attention 0, mlp0, and mlp1. We sample each of these to be from a separate sequence, where all three agree on $horizon_i$. The rest of the sum (attention 1 and the embeddings) will thus be sampled on a random input.



Surprisingly, this causes the loss recovered by the scrubbed model to improve significantly when compared to experiment 3a, to 84%. Why is this? Shouldn't claiming a more specific hypothesis result in lower loss recovered?

We saw in experiment 3a that certain inputs which pass the horizon test at i still carry unbalanced-signal within $x_2[i]$. However, instead of sampling a single input for $x_2[i]$ we now are sampling four different inputs: one each for a0, mlp0, and mlp1 which all agree on the horizon $_i$ test, and a final random input for both the embedding and a1. Sampling the terms of $x_2[i]$ is enough to 'wash out' this unbalanced signal in some cases.

In fact, it is sufficient to just sample $x_2[i]$ as the sum of two terms. Consider the intermediate hypothesis that all that matters is the *sum* of the outputs of a0, mlp0, and mlp1:



By not including the emb + a1 term, this hypothesis implicitly causes them to be sampled from a separate random input.

The % loss recovered is 81%, between that of 3a and 3c. As a summary, the following table shows which terms are sampled together:

3a (1 term)	$x_2[i]$ = emb + a0 + mlp0 + a1 + mlp1	%LR = 77%
3d (2 terms)	$x_2[i]$ = emb + a0 + mlp0 + a1 + mlp1	%LR = 81%
3c (4 terms)	$x_2[i]$ = emb + a0 + mlp0 + a1 + mlp1	%LR = 84%

In red are the claimed-unimportant terms, sampled from a random input. All other inputs agree with horizon $_i$. Note also that in each case, all inputs are independently drawn between positions.

Are the results of experiment 3c and 3d legitimate then? We think not. One way to think about this is that a hypothesis makes claims about what scrubbing should be legal. For instance, the hypothesis in 3d claims that it would be okay to sample $x_2[i]$ separately, term by term. However, the hypothesis also implies that it would be okay to sample them together!

One way to address this sort of problem is to introduce an adversary, who can request that terms are sampled together (if this is allowable by the proposed hypothesis). The adversary would then request that the hypotheses from 3c and 3d are run with every term of $x_2[i]$ sampled together. This would result in the same experiment as we ran in 3a.

1. ^

   In most of the interpretability research we're currently doing, we focus on tasks that a simple algorithm can solve to make it easier to reason about the computation implemented by the model. We wanted to isolate the task of finding the clearest and most complete explanation as possible, and the task of validating it carefully. We believe this is a useful step towards understanding models that perform complex tasks; that said, interpretation of large language models involves additional challenges that will need to be surmounted.

2. ^

   We limit to just the random dataset mostly to make our lives easier. In general, it is also easier to explain a behavior that the model in fact has. Since the model struggles on the adversarial datasets, it would be significantly more difficult to explain 'low loss on the full training distribution' than 'low loss on the random subset of the training distribution.'
   It would be cleaner if the model was also exclusively trained on the random dataset. If redoing our investigation of this model, we would train it only on the random dataset.

3. ^

   In particular, we can write the output of the model as $f(x_{2.0}+y)$, where f is the final layer norm and linear layer which outputs the log-probability that the input is balanced, $x_{2.0}[0]$ is the output of head 2.0 at position 0, and y[0] is the sum of all other terms in the residual stream. Then we compute the attribution for 2.0 as $E_{y'}[f(x_{2.0} + y')]$ where y' is sampled by computing the sum of other terms on a random dataset sample. We do the same to get an attribution score for head 2.1. Other attribution methods such as linearizing layer norms give similar results.

4. ^

   The same algorithm is applied to a small language model on induction here, but keep in mind that some conventions in the notation are different. For example, in this post we more explicitly express our interpretation as a computational graph, while in that post we mostly hypothesize which paths in the treeified model are important; both are valid ways to express claims about what scrubs you ought to be able to perform without hurting the performance of the model too much. Additionally, since our hypothesis is that important inputs need not be equal, our treeified model is run on many more distinct inputs.

5. ^

   "Direct connection" meaning the path through the residual stream, not passing through any other attention heads or MLPs.

6. ^

   These will be inputs like `)(()` with equal amounts of open and close parentheses, but a close parentheses first. $x + 1$

7. ^

   One technique that can be helpful to discover these sorts of problems is to perform a pair of experiments where the only difference is if a particular component is scrubbed or not. This is a way to tell which scrubbed inputs were especially harmful – for instance, the fails-count-test inputs being used for 2.0 or 1.0 hurting the loss in 1a.

8. ^

   We exclude paths through attention layer 0 when creating the indirect emb node

9. ^

   We originally theorized this by performing an “minimal-patching experiment” where we only patched a single sequence position at a time and looking for patterns in the set of input datapoints that caused the scrubbed model to get high loss. In general this can be a useful technique to understand the flaws of a proposed hypothesis. Adding this fact to our hypothesis decreased our loss by about 2 SE.

10. ^

    This does ignore attention on the [BEGIN] and [END] positions, but in practice this doesn’t change the output noticeably.

11. ^

    Using the adjusted horizon test in Experiment 1b slightly increases the loss to 0.33, not a significant difference. It is perhaps somewhat surprising the loss doesn’t decrease. In particular we should see some improvement when we wanted to sample an output from 2.0 and 1.0 that passes the count(test, but an output from 2.1 that fails the horizon test, as we no longer sample false-negatives for 2.1 (where the output has no unbalanced evidence). This is a rare scenario, however: there aren’t many inputs in our dataset that are horizon failures but pass the count(test. We hypothesize that this is why the improvement doesn’t appear in our overall loss.