Gradient Descent Over Interpolated Activation Patches for Circuit Discovery

Glen M. Taggart

In gpt-2-small there are $h \sum_{l=0}^{n-1} lh = n(n-1)/2 \cdot h^2 = 9504$ "edges" between attention heads and in general $edges = O\left(h \sum_{l=0}^{n-1} lh\right) = O\left(n^2 h^2\right)$ edges available to be patched in circuit discovery. It is intractible to patch all combinations of edges, as that is

$$O\left(2^{edges}\right) = O\left(2^{(n^2 h^2)}\right)$$

or in this case 2^{9504} combinations to check. So, we are interested to find robust computational approaches that do not experience this combinatorial explosion. The approach implemented in this project works assigns learnable patch interpolation coefficients to each of the edges (as well as connections from the corrupted residual stream to the attention heads and from the head to the output), and then does gradient descent on these coefficients.

Edges are of the form: (src layer, src head)→(dest layer, dest head)

Algorithm 1: Unoptimized psuedocode for gpt-2 (with 12 layers and 12 heads). Glosses over the output coefficients and residual input coefficient.

Input : input
θ: parameters; edge weight coefficients as $\theta_{src,dest}$
C: head activations from a clean forward pass
`forward()` : function; normal model forward to get cached values for heads
`forwardPatched(`*input*`){`$head_{lh} \leftarrow \ldots$`}` : forward pass with head h of layer l patched. Returns logits.
`forwardPatched[i, j](`*input*`){`$head_{lh} \leftarrow \ldots$`}` : forward pass with head h of layer l patched. Returns head j of layer i.

Output: out: patched logits to perform gradient descent on

```
for i ∈ {1, 2, ..., 12} do
    for j ∈ {1, 2, ..., 12} do
        patched_ij ← forwardPatched[i, j](input) {
            for l ∈ {1, 2, ..., i − 1} do
                for h ∈ {1, 2, ..., 12} do
                    head_lh ← C_ij · (1 − θ_lh,ij) + patched_lh · θ_lh,ij
                end
            end
        }
    end
end
out ← forwardPatched(input) {
    for l ∈ {1, 2, ..., 12} do
        for h ∈ {1, 2, ..., 12} do
            head_lh ← patched_lh
        end
    end
}
```

then out is used to calculate a loss and gradient descent can be performed over θ

The naive implementation of this requires something in the range of $O(n^2h^2)$ to $O(nh)$ (depending on how naive) language model forward passes. However you can do parallel src-heads within the same layer together by stacking them into a batch dimension (moving that bit of complexity from time to space), and you don't need to recalculate all heads in earlier layers at each layer, which brings it down to $O(n)$ forward passes.

My plan was to start by implementing this just to clarify some things for myself, then have it require an utterly unworkable amount of vram, after which I would start by modifying the technique to first prune the graph by learning (head,layer)->(layer) edges prior to doing full edge processing on the subgraph produced by that process. However, it's actually just fully runnable currently! I still think

these improvements would be a good idea, as running a single iteration takes 10-30 seconds and at a batch size of 8 it uses ~15gb of VRAM. It's very slow, too, as one would expect for something that requires $O(n)$ forward passes of gpt2 per optimization step...

Nevertheless, it works somewhat at finding IOI edges! and manages to fit in VRAM!

It recovers some of the IOI circuit along with some false positives, I haven't gotten to hyperparameter tune it enough to fully assess if this is capable of doing a decent job discovering circuits. Current performance is not competitive.

It seems semi possible it could have some advantages over attribution patching if I work out the kinks (if it could run fast?), but the limitations (sxlow, complex) may be pretty hard to get past and the scaling properties are not great without some preprocessing to prune the graph. The main advantage is that the gradients are (closer to) in the context of the learned circuit, which is an important feature when there are nonlinear dependencies between circuit components. Such dependencies can also create problems for this method from what I've observed, but there are also things I can imagine to do to mitigate this.

This is still very much a work in progress and there are many features I still want to add, including:

- patching in residual stream values by token position
- adversarial coefficients network so that it discovers negative name mover heads
- patching separately to q, k, v
- pruning modes
  - mode where all nodes are also connected not just to children but to all descendants to make the time complexity O(hn)
  - edges with layer destinations only (this is implemented but needs to be optimized for it to be worth it)
  - ignore mlps so you can do it in just one gpt2 forward pass?

sample from the model working on IOI:

```
0.10 <- RS_0
2.2 <- 0.1,
2.11 <- 0.10,
3.0 <- RS_0 | <-0.1, 0.9, 2.9,
4.7 <- 0.1, 0.7, 0.10, 1.11, 2.2, 3.7,
5.5 <- RS_0 | <-0.1, 1.11, 3.0, 4.7,
6.1 <- 1.11, 3.0, 5.5,
6.7 <- 5.5,
6.8 <- 5.5,
6.9 <- RS_0 | <-0.1, 3.0,
7.1 <- 5.5,
7.9 <- 0.1, 0.10, 2.11, 3.0, 5.5, 6.9,
```

```
8.6 <- 3.0, 5.5, 6.9, 7.9,
8.10 <- 0.1, 3.0, 5.5, 6.9,
9.6 <- 0.10, 3.0, 5.5, 6.1, 7.9, 8.6, 8.10,
9.9 <- 0.1, 3.0, 5.5, 6.1, 7.9, 8.6, 8.10,
10.0 <- 0.10, 1.11, 3.0, 4.7, 5.5, 6.1, 6.9, 7.9, 8.6, 8.10,
10.1 <- 8.10,
10.10 <- 7.9, 8.6, 8.10,
11.2 <- 0.10, 1.11, 3.0, 5.5, 6.1, 8.6, 8.10,
11.6 <- 3.0, 7.9, 8.6,
11.10 <- 10.7,

| 9.6, 9.9, 10.0, 10.10, -> out
```