6.7960 Project Proposal: Investigating Off-Distribution Generalization of Transformers

Alek Westover, Anthony Wang, Kevin Zhao

In this project, we hope to further our understanding of *when* off-distribution generalization happens. Paul Christiano proposed an experiment here about shortest paths in a graph to investigate this; our project is essentially to implement Christiano's proposed experiment. To the best of our knowledge, no one has done this yet.

Motivation

It is generally desirable for LLMs to output true statements. One current approach for ensuring this is to have a human in the loop rewarding the model for true outputs (e.g. RLHF). One drawback of this approach is that humans can be poor judges of truthfulness. Humans enjoy many cognitive biases and might employ superficial heuristics when judging truthfulness. A further challenge is that as LLMs become more capable, there might not even exist experts that are good judges of whether the models outputs, such as difficult mathematical proofs, are truthful.

One approach to solving this problem is to reward an LLM for truthful behavior on simple inputs, and then hope that the LLM generalizes its truthful behavior for more complex inputs where humans cannot provide helpful labels. Deep learning models often perform remarkable feats of off-distribution generalization – for instance, a model trained to transform hand drawn cats into images of cats might be able to handle a "cat" with three eyes in an intuitive way. We might hope that generalizing truthfully is simple, thus promoted by "Occam's Razor", and aim to investigate that with this project.

Task

We will use a synthetic task to test our hypothesis that models will generalize truthfully off-distribution. The synthetic task is computing the distance between various vertices in an input graph. Our experiment will have three parts:

1. Pre-train a transformer to predict the distance between two fixed vertices s, t on graphs with $n \in [8, 32)$ vertices.
2. Fine-tune a transformer to predict the distances between s, t' for any t' which is on the shortest path from s to t, but only do fine-tuning on graphs with $n \in [8, 16)$ vertices.
3. Test whether the transformer can accurately predict the distances between s, t' for any t' on the shortest path from s to t for graphs with $n \in [16, 32)$ vertices.

Data

We will represent an n vertex, m edge unweighted, undirected graph as sequence of the endpoints of the m edges, so $[a_1, b_1, a_2, b_2, ..., a_m, b_m]$ represents a graph with the edges $\{(a_i, b_i)\}$ for $1 \leq i \leq m$. We will pad all sequences to be the same length using the padding token 0.

The full input to our model will additionally add the target vertex after the padding tokens. The model is tasked with predicting the length of the shortest path between vertex 1 and the target vertex t. If no such path exists, we define the length to be $n + 1$ which represents infinity. For example, an input-output pair for our model could look like $[1, 3, 3, 2, 0, 0, 0, 0, 2]$ and 2 respectively.

We have three separate datasets.

- **Pre-train data**: For each $n \in [8, 32)$, we will generate several graphs on n vertices. We generate these graphs by inserting $2n$ random edges into the graph. We always set the target vertex to be 2 here.
- **Fine-tune data**: For each $n \in [8, 16)$, we will generate several graphs on n vertices. We generate these graphs by inserting $2n$ random edges into the graph. We select the target vertex to be a random vertex on the shortest path from 1 to 2.
- **Generalization testing data**: The same as the fine-tune data, except we sample $n \in [16, 32)$ instead.

As a side note, we are also curious whether the transformer learns to generalize to different distributions of graphs, such as denser graphs or graphs with different properties. Time permitting, we will also investigate this.

Architecture

We plan to use a standard transformer architecture. We will ensure that the number of layers in our transformer is at least the diameter of the graph. By doing this, we ensure that there is an extremely simple circuit — namely BFS — that the transformer could in theory learn to perform the task. Note that if the transformer actually learns a simple circuit to perform this task, then it seems more likely to generalize well. This is also our intuition for why it should be possible to fine tune on a small amount of data for finding shortest paths to other vertices besides 2 – it seems like the model should be computing these other distances as intermediate values in its computation to find the distance to vertex 2.

Positional Encodings

In order to facilitate performing this task with limited computational resources, we plan to use custom-made positional encodings that tell the model extra information about the structure of the problem, rather than the traditional sine/cosine positional encodings. Specifically, our positional encodings are $v_1, v_1, v_2, v_2, ..., v_m, v_m, v_{m+1}$ where each v_i is a random vector so each v_i, v_j pair is nearly orthogonal with high probability. We will concatenate these with the token encodings rather than adding them. This should let the model easily have large attention scores between vertices corresponding to a single edge.