Brain Surgery

MCDA5511 Assignment #4

Due: Apr 4 before midnight

Set Up

In this assignment you will build a toy version of Anthropic's sparse autoencoder for explaining Large Language Models. Begin by familiarizing yourself with the paper *Mapping the Mind of a Large Language Model* (May 2024), alternatively titled *Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet*, available here: https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html

Anthropic's achievement with this paper was to deploy these techniques at scale for a state-of-the-art foundation model. For our purposes, we will try to replicate their results as closely as possible with a smaller LLM. You are free to choose whichever model you wish. You can complete the assignment on a CPU with GPT-2, or even a model you train yourself, but higher marks will be awarded for larger models using whatever compute you have at your disposal.

Homework

For this assignment, you will use GitHub to submit your homework.

- **DO NOT use Jupyter notebooks for this assignment.** Instead, create a codebase using the best practices you learned in the tutorial, with an *environment.yml* file, a *main.py* file, and any other supporting files you need.
- Submit your repo for grading by inviting the instructors to collaborate on the project in GitHub when you are finished.
- Same as with previous assignments documented using GitHub, write up the answers to questions in your *readme* in the main branch of your repo.

(1) Begin by selecting a model and building a wrapper class that handles text generation with activation collection. It is wise to start with a small model and swap it out with a more compute-intensive model later when you are confident that your code works. You should always write your code in a sufficiently modular way to allow for these sorts of changes.

Your class will need the following components:

- A method to intercept the forward pass and collect the activations. Note that PyTorch has built in functionality called "hooks" that do this.
- A method to generate text and return it along with the activations.

In your *readme*:

- Explain your reasoning for why you inserted hooks where you did. The Anthropic paper provides reasons to insert the hooks in a middle layer residual stream. If you decide to follow them, you can just explain their rationale in your own words, just remember to cite your sources. What are the limitations of your selected approach in terms of being able to truly explain the behaviour of the LLM?

(2) Next, build a class that generates training data for the autoencoder. Your class will need the following components:

- A method to iteratively work through a corpus of prompts, calling your model to generate text and activations for each prompt.
- A method to save the outputs in a format that can be input to the autoencoder for training. Note that you will need to retain mappings from text/tokens to activations for interpretation later.

Download or generate a corpus of prompts and use them to create your training dataset. In your *readme*, show some summary statistics describing your data and explain why you chose the prompts that you did. For example, are you trying to focus on a specific topic of interest?

(3) Now you can train your autoencoder. Start by familiarizing yourself with the sparse autoencoder structure by reviewing Cunningham, Hoagy, et al. "Sparse autoencoders find highly interpretable features in language models." *arXiv preprint arXiv:2309.08600* (2023). Then write an autoencoder class with the following components:
- Autoencoder structure: Encoder-decoder, tied weights, L1 regularization to enforce sparsity.
- You may wish to fix key hyperparameters such as the L1 penalty and encoder dimension for now, and experiment with different values later to see if they give you better results for the remaining questions. Document your hyperparameter setting and experiments in your *readme*. Note that the input and output dimension is your activation dimension, and the encoder dimension should be larger than the activation dimension (start by choosing something 2 – 4x larger and adjust later as needed).
- Remember to use early stopping and track the training metrics to make sure the autoencoder is learning. Document your results in your *readme*.

In your *readme*, explain in your own words what the autoencoder is doing in activation space. Start by explaining what is meant by "activation space." How does this relate to the space of "features" (in Anthropic's sense of the term)? Show that your autoencoder has in fact learned a sparse representation.

(4) With all that done, you can now analyze the data you created to see if you can explain the behaviour of the LLM. Effectively, you now want to run the process from the previous questions backwards: Pick a dimension from your encoder, map that back to the activations that fired strongly for it, and then map those back to the text snippets that caused the activations. You can then manually inspect the text snippets to see if you can determine what they have in common. This is the interpretable "feature." For example, the first features mentioned in the Anthropic paper are "sycophantic praise" and "the Golden Gate bridge."

For this question, identify features manually. In your *readme*, provide 3 – 5 examples of features that you analyzed, with their text snippets, and your interpretation of what feature they indicate. If your text snippets are not interpretable (i.e. you can't think of what feature they have in common), go back and adjust your model pipeline (e.g. hyperparameters, text examples, other aspects of the code you wrote) to see if you can get it to produce more easily interpretable results. If that still doesn't work, simply state that you were not able to produce interpretable features.

(5) Optional bonus question: If you wish, you can experiment with automated interpretability techniques. Basically, this means using ML techniques to identify features instead of manually inspecting text snippets. For example, you could look at correlations in your feature space (the space that your sparse autoencoder has learned) and cluster them to look for related features. You could also query an LLM to ask what a given set of text snippets have in common (i.e. have the LLM identify what the interpretable feature is). The goal here is to be able to draw conclusions about the full set of features, rather than hand picking examples.

(6) To prove that your code works, select a feature that you identified and clamp it (i.e. intervene in the neural network to ensure that the neurons your autoencoder has associated with the feature are always activated). Use this to perform counterfactual experiments where you compare behaviour between the baseline (unclamped) and clamped versions of the model. Refer to the Anthropic paper for examples to guide you.