DS-GA-1011: Natural Language Processing with Representation Learning, Fall 2024

HW-4: Scaling Language Models and Prompt Engineering

Name
NYU ID

Please write down any collaborators, AI tools (ChatGPT, Copliot, codex, etc.), and external resources you used for this assignment here.
Collaborators:
AI tools:
Resources:

By turning in this assignment, I agree by the honor code of the College of Arts and Science at New York University and declare that all of this is my own work.

Acknowledgement: Xiang Pan set up the leaderboard and Lavender Jiang developed the programming question. We drew inspiration from the scratchpad paper by Nye et al. and prompt engineering guide. Ryan Teehan contributed question 2.

Before you get started, please read the Submission section thoroughly.

Submission

Submission is done on Gradescope.

Written: You can either directly type your solution in the released `.tex` file, or write your solution using pen or stylus. A `.pdf` file must be submitted.

Programming: Questions marked with "coding" at the start of the question require a coding part. You should submit `submission.py`.

Due Date: This homework is due on November 13, 2024, at 12 pm Eastern Time.

1 Language Models and Compression

In this problem, we will try to connect language models to lossless compression. The goal of lossless compression is to encode a sequence of symbols $x = (x_1, \ldots, x_n)$ $(x_i \in \mathcal{X})$ following a distribution p to a sequence of bits, i.e. $c : \mathcal{X}^* \to \{0, 1\}^*$, such that the original sequence can be recovered from the bit sequence. To increase compression efficiency, we want to minimize the expected number of bits per sequence.
Shannon's source coding theorem states that the minimum number of bits (using any compression algorithm) cannot go below the Shannon entropy of the sequence $H(X) = \mathbb{E}_{x\sim p}[-\log_2 p(x)]$ (note that here we use X to denote the random variable and x to denote the value).

1. [2 points] Now, given a language model q trained on a finite sample of sequences from p. Show that the perplexity of the language model is lower bounded by $2^{H(p)}$:

$$2^{\mathbb{E}_{x\sim p}[-\log_2 q(x)]} \geq 2^{H(p)}$$

[**HINT:** You can use the fact that the KL divergence is non-negative: $D_{\text{KL}}(p\|q) \geq 0$.]

2. Recall that in Huffman coding, we construct the Huffman tree based on frequencies of each symbol, and assigning binary codes to each symbol by travesing the tree from the root. For example, given a set of symbols $\{a, b, c\}$ and their corresponding counts $\{5, 10, 2\}$, the codeword for a, b, c are 01, 1, 00, respectively. You may want to review Huffman coding at `https://en.wikipedia.org/wiki/Huffman_coding`.

   Note that instead of using counts/frequencies to construct the tree, we can use any weight proportional to the probability of the symbol, e.g., 5/17, 10/17, 2/17 in the above example. Now we will use a toy corpus to estimate the probabilities of each symbol, derive the Huffman code from the probabilities, and encode a sequence.

   (a) [2 points] Given the following corpus (you can assume each token is separated by a whitespace):

   ```
   the cat was on the mat .
   the cat on the mat has a hat .
   the mat was flat .
   ```

   Estimate the unigram probability of each token in the vocabulary.

3. [3 points] Use the above probabilities to construct a Huffman tree for the symbols, and encode the sequence `the mat has a hat` and `the hat has a mat`.

4. [4 points] Note that in the above question, the two sequences have the same code length because we are encoding each word independently. In principle, we should be able to shorten the code length by considering dependencies between a word and its prefix. Specifically, given a language model q, for the first word in a sequence, we encode it by constructing a Huffman tree using weights $q(\cdot)$; for the i-th word, we encode it using weights $q(\cdot \mid x_{<i})$. Now, encode the sequence `the mat has a hat` using a bigram language model estimated on the above toy corpus. [**NOTE:** Because we are not using any smoothing, $q(\cdot \mid x_{<i})$ assigns zero probability to certain tokens, which means that we won't be able to encode all possible sequences. But you can ignore this problem in the context of this question.]

2 Model Stealing

In this problem, we will use a mathematical treatment of prompting to extract information from a language model. For this problem, assume we have a language model $\mathcal{M}_\theta(p)$, which maps from an input prompt p to a probability distribution $\mathcal{P}(\mathcal{X})$ over a vocabulary $\mathcal{X}$, where $p \in \mathcal{X}^N$ for a prompt of length N. Given a parameterized *trunk* $g_\theta(p)$ which produces the final hidden state, we can write $\mathcal{M}_\theta(p)$ as:

$$\mathcal{M}_\theta(p) = \text{softmax}(\mathbf{W} \cdot g_\theta(p)) \tag{1}$$

where $\mathbf{W} \cdot g_\theta(p) \in \mathbb{R}^\ell$ are the *logits* and $\mathbf{W}$ is the final projection matrix. If the vocabulary size, $|\mathcal{X}|$, is ℓ and the hidden dimension is h, then $\mathbf{W}$ has dimensions $\ell \times h$. Assume that we can directly observe the logits by querying an oracle $\mathcal{O}(p)$, that $h \ll \ell$, and that the weight and activation matrices are full-rank.

1. [2 points] Design an algorithm which allows you to recover the hidden dimension h repeatedly querying the oracle $\mathcal{O}(p)$ for logits. Show that your approach will correctly recover the hidden dimension. [**HINT:** Consider the rank of a matrix of logits]

2. [2 points] Building on your previous algorithm, show how we can recover the final projection matrix $\mathbf{W}$ up to linear transformation. That is, show that we can recover $\tilde{\mathbf{W}} = \mathbf{W} \cdot \mathbf{G}$ for some $\mathbf{G} \in \mathbb{R}^{h \times h}$.

3 Prompt Engineering for Addition

The goal of this coding problem is the following:

1. Give you hands-on experience with prompt engineering.
2. Understand the challenges involved in prompt engineering.

Specifically, we will use API from together.ai to teach the LLaMa-2 7B model to add two positive 7-digit integers.
First go through the file `README.md` to set up the environment required for the class.

1. Zero-shot Addition

We provided you a notebook `addition_prompting.ipynb`. In the first section, there are two examples of zero-shot addition: ten 1-digit addition and ten 7-digit additions.

a. (2 points, written) Run the two examples. In your opinion, what are some factors that cause language model performance to deteriorate from 1 digit to 7 digits?

b. (5 points, written) Play around with the config parameters in together.ai's web UI.

- What does each parameter represent?
- How does increasing each parameter change the generation?

c. (2 points, written) Do 7-digit addition with 70B parameter llama model.

- How does the performance change?
- What are some factors that cause this change?

d. (2 points, written) Previously we gave our language model the prior that the sum of two 7-digit numbers must have a maximum of 8 digits (by setting `max_token=8`). What if we remove this prior by increasing the `max_token` to 20?

   - Does the model still perform well?
   - What are some reasons why?

2. In Context Learning

In this part We will try to improve the performance of 7-digit addition via in-context learning. For cost-control purposes (you only have $25 free credits), we will use llama-2-7b.

a. (2 points, written) Using the baseline prompt ("Question: What is 3+7? Answer: 10 Question: What is a+b? Answer:"), check 7-digit addition for 10 pairs again.

- Compared to zero-shot 7-digit additions with maximum 7 tokens, how does the performance change when we use the baseline in-context learning prompt?
- What are some factors that cause this change?

b. (3 points, written) Now we will remove the prior on output length and re-evaluate the performance of our baseline one-shot learning prompt. We need to modify our post processing function to extract the answer from the output sequence.

- Describe an approach to modify the post processing function.
- Compared to 2a, How does the performance change when we relax the output length constraint?
- What are some factors that cause this change?

c. (4 points, written) Let's change our one-shot learning example to something more "in-distribution". Previously we were using 1-digit addition as an example.
Let's change it to 7-digit addition (`1234567+1234567=2469134`).

- Evaluate the performance with max_tokens = 8. Report the `res, acc, mae, prompt_length`.
- Evaluate the performance with max_tokens = 50. Report the `res, acc, mae, prompt_length`.
- How does the performance change from 1-digit example to 7-digit example?
- Take a closer look at `test_range`. How was `res` calculated? What is its range? Does it make sense to you? Why or why not?

d. (written, 4 points) Let's look at a specific example with large absolute error.

- Run the cell at least 5 times. Does the error change with each time? Why?
- Can you think of a prompt to reduce the error?
- Why do you think it would work?
- Does it work in practice? Why or why not?

3. Prompt-a-thon!

In this part, you will compete with your classmates to see who is best at teach llama to add 7-digit numbers reliably! Submit[1] your `submission.py` to enter the leader board!

The autograder will test your prompt on 30 pairs of 7-digit integer addition. Some pairs are manually designed and some pairs are randomly generated using a random seed that is different from `addition_prompting.ipynb`. This will generate 30 API calls, with 1 second wait time between each API call. Since prompting has randomness, we will run the same trial 3 times and report the average performance. In total, each submission will incur 90 API calls.

Please keep `max_tokens` to at least 50, so that you're not giving the model a prior on the output length. Please also do not try to hack the pre-processing and post-processing functions by including arithmetic operations. The autograder is able to detect the use of arithmetic operations in these functions.
Here's a list of functions you can modify:

- `your_api_key`
- `your_prompt`
- `your_config`
- `your_pre_processing` (but no arithmetic operation. e.g., `f"a+b={a+b}"` is not allowed.)
- `your_post_processing` (but no arithmetic operation. e.g., `f"a+b={a+b}"` is not allowed.)

a. (coding, 5 points) Use prompt to improve test performance. For full credit, either

   - get average accuracy greater than 0.1, or
   - get average mean absolute error less than 5e6.

   To prevent recovery of test pairs, your autograder submission is limited to 10 tries. For testing, we recommend using `test_prompts.py`.

b. (optional, 1-3 points) Top 3 students on the leaderboard will be given the following awards:

   (a) First place: 3 points
   (b) Second place: 2 points
   (c) Third place: 1 point

   We will add the points to your homework 4 score (capping at the maximum). For example, suppose the student in first place originally has 23 out of 25, with the reward they will get 25 out of 25. Suppose the student in first place originally has 20 out of 25, with the reward they will get 23 out of 25. In case of a tie, we recognize the earlier submission.

[1] Note: while you can use `prompt.txt` for debugging and local testing, for the final autograder submission, please use string (**not file**). The reason why is that autograder cannot find `prompt.txt` in the testing environment. Sorry about the inconvenience!