CS 601.471/671 NLP: Self-supervised Models

Homework 7: Prompting Language Models + Retrieval-Augmented LMs

For homework deadline and collaboration policy, check the calendar on the course website[*]

Name: ____________________

Collaborators, if any: ____

Sources used for your homework, if any: ______________________________

This assignment is testing your ability to work with language models, particularly focusing on in-context learning and retrieval-augmented language models.

Note: You can do this homework as a team of 2 people. Basically, you can write a single solution for both people and upload it as a single PDF as a group.

How to hand in your written work: Via Gradescope as before.

Homework 7 is here, it's time to explore
The world of language models, we'll learn to adore
Prompting and retrieval-augmented, we'll dive deep
Into the intricacies of language, we'll take a leap.

We'll start with the basics, what are prompts you ask
A few words or phrases, a task to unmask
The true meaning of text, to generate new
Language that's coherent, and insightful too.

Next, we'll move to retrieval, a powerful tool
To find the right text, and not play the fool
By using context, we'll get what we need
To create language models that truly succeed.

So let's get to work, on this homework seven
With prompting and retrieval, we'll reach language heaven
We'll build models that are efficient and precise
And create language that's both eloquent and nice.

So let's not delay, let's dive right in
To the world of language, where we'll learn to win
With prompting and retrieval, we'll make our mark
On the world of language models, it's time to embark.
–ChatGPT March 11 2023

1 Concepts, intuitions and big picture

1. Which one(s) are valid motivations for natural language prompting of LMs instead of fine-tuning them?
☐ Prompting results in multiple copies of model parameters for each training task, which creates redundancy.
☐ Fine-tuning results in multiple copies of model parameters for each training task, which creates redundancy.
☐ Prompts are more interpretable.
☐ Fine-tuned parameters are more interpretable.
☐ Prompting requires backward computation of gradients.
☐ Fine-tuning requires backward computation of gradients.
Answer: *TBD*

2. Which one(s) are correct about natural language prompts?
☐ Models interpret prompts like humans do.
☐ Models are sensitive to the ordering of the instances matter.
☐ Models are sensitive to the phrasing of prompts.
☐ Models have recency bias – labels that appear later tend to get more probability mass.
☐ Models may be biased to generating popular outputs. Hence, calibrating them is a necessary step.
☐ With scale, performance *always* increases.
☐ With scale, performance *generally* increases.
☐ Discrete prompts are a concrete step in progress toward robust models of human-machine communication.
Answer: *TBD*

[*] https://self-supervised.cs.jhu.edu/sp2023/

2 Revisiting Self-Attention: Sensitivity of Attention Layers

In this homework we will quantify the sensitivity of an attention layer with respect to perturbations in its input. Assume that, we are given a sequence of input embeddings $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ where $\mathbf{x}_t \in \mathbb{R}^d$. For each position t, we have key, query and values:

$$\mathbf{q}_t = \mathbf{W}_q \mathbf{x}_t, \;\; \mathbf{k}_t = \mathbf{W}_k \mathbf{x}_t, \;\; \mathbf{v}_t = \mathbf{W}_v \mathbf{x}_t$$

Then the unnormalized attention weights (the amount that position i attends to position j before doing Softmax) are $s_{ij} = \mathbf{q}_i \mathbf{k}_j^\top$. Then, the attention outputs at position t is:

$$\mathbf{a}_t = \text{Softmax}\left(\frac{[s_{t1}, s_{t2}, \ldots, s_{tn}]}{\sqrt{d}}\right) [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]^\top \in \mathbb{R}^d$$

Now consider a small perturbation added to the input at position $\hat{t}$: $\mathbf{x}_{\hat{t}} \to \mathbf{x}_{\hat{t}} + \Delta$ while all the other positions $t \neq \hat{t}$ are unchanged. We would like to measure the effect of this input perturbation on the output of the attention mechanism.

1. Given a perturbation of an input at position $\hat{t}$, the attention weights would also change. Let $\hat{s}_{ij}$ denote the unnormalized attention weights after the perturbation. Then prove that for any $i \neq j$:

$$\hat{s}_{ij} = \begin{cases} s_{ij} & j \neq \hat{t} \\ s_{ij} + \mathbf{q}_i (\mathbf{W}_k \Delta)^\top & j = \hat{t} \end{cases}$$

   Answer: *TBD*

2. Assume that, the perturbation vector has a bounded norm: $\|\Delta\| \leq \delta$ (i.e., the unit sphere) for some small positive scalar value of δ. Also assume that the perturbed input embeddings are uniformly distributed on the unit sphere. Then prove that, the expected change of the attention weights are bounded. Specifically:

$$\mathbf{E}[|\hat{s}_{ij} - s_{ij}|] \leq \frac{C\delta}{\sqrt{d}}$$

   with $C = \|\mathbf{W}_q\| \, \|\mathbf{W}_k\|$.
   Answer: *TBD*

3. **Extra Credit:** Probability that the attention weights are larger than a given ϵ is small:

$$\mathbf{P}(|\hat{s}_{ij} - s_{ij}| \geq \epsilon) \leq \frac{C\delta}{\epsilon\sqrt{d}}$$

   Hint: Use Markov inequality.
   Answer: *TBD*

3 How to Sum like Einstein (Einstein Summation Notation)

Einstein summation is a convention for simplifying expression that includes summation of vectors, matrices or in general tensor. This convention allows a compact statement of lengthy summation operators. At times, using this operator can be even faster than the usual way of implementing summations since PyTorch may execute the summations in a way that is more optimized.

Take this example: Matrix multiplication of `A` and `B` can be computed using `einsum` as `torch.einsum("ij,jk->ik", A, B)`. Here, `j` is the summation subscript and `i` and `k` the output subscripts. Note, the equation may contain white-spaces between the different elements (subscripts, arrow and comma). There are additional nuances in the notation (e.g., ellipsis `...`) which can be found in the definition of `einsum(.)`.

Since `einsum(.)` is used frequently in PyTorch implementations and its interpretation can be somewhat cryptic, here we want to challenge ourselves with examples of this operator. Last but not least, this operator also exists in NumPy with slight differences with its PyTorch implementation.

1. Assume that `u` is given with 2 rows and 3 columns.

```
import numpy as np

u = np.full((2,3),2)
```

1.1. Use `np.einsum(.)` operator to sum along the columns. In this case output will be a 3 dimensional vector.

Your Response

```
TODO
```

1.2. Use `np.einsum(.)` operator to sum along the rows.

Your Response

```
TODO
```

2. Define the following variables.

```
import torch

x = torch.rand(4, 4)
y0 = torch.rand(5)
y1 = torch.rand(4)
z0 = torch.rand(3, 2, 5)
z1 = torch.rand(3, 5, 4)
w = torch.rand(2, 3, 4, 5)
r0 = torch.rand(2, 5)
r1 = torch.rand(3, 5, 4)
r2 = torch.rand(2, 4)
s0 = torch.rand(2, 3, 5, 7)
s1 = torch.rand(11, 3, 17, 5)
```

Now, interpret the provided expressions with `np.einsum(.)` operator and rewrite them without this operator.

```
# identity
a0 = torch.einsum('i', y0)
a1 = torch.einsum('ij', x)
a2 = torch.einsum('ijk', z0)

# permute
b0 = torch.einsum('ij->ji', x)
b1 = torch.einsum('ba', x)
b2 = torch.einsum('jki', z0)
b3 = torch.einsum('ijk->kij', z0)
b4 = torch.einsum('kjil', w)
b5 = torch.einsum('...ij->...ji', w)
b6 = torch.einsum('abc...->cba...', w)
```

```
# trace
c = torch.einsum('ii', x)

# sum
d0 = torch.einsum('ij->', x)
d1 = torch.einsum('xyz->', z0)
d2 = torch.einsum('ijkl->', w)

# sum axis
e0 = torch.einsum('ijk->i', z0)
e1 = torch.einsum('ijk->j', z0)
e2 = torch.einsum('ijk->ij', z0)

 # matrix-vector
f0 = torch.einsum('ij,j->i', r0, y0)
f1 = torch.einsum('i,jki->jk', y1, r1)

# vector-vector outer product
g0 = torch.einsum('i,j->ij', y0, y1)
g1 = torch.einsum('a,b,c,d->abcd', y0, y1, y0, y1)

# batch mm
h0 = torch.einsum('bij,bjk->bik', z0, z1)
h1 = torch.einsum('bjk,bij->bik', z1, z0)

# bilinear
i = torch.einsum('bn,anm,bm->ba', r0, r1, r2)

# tensor contraction
j = torch.einsum('pqrs,tqvr->pstv', s0, s1)
```

Your Response

```
TODO
```

3. Look at the following usages of `np.einsum(.)` operator. For each instance, explain briefly (no more than 5 sentences for each use case) what the operator is doing.

   - Use case 1
   - Use case 2

   Answer: *TODO*

4 EC: Retrieval Augmented Language Models

In this homework we want to repeat HW6 (BoolQ) with a retrieval-augmented language model. Specifically, we will use RAG. Train a yes/no classifier using this language model such that given a question, it would map it to yes/no answers. Compare the performance of the model in two scenarios: (1) the default setting where a question comes with its gold paragraph, vs. (2) questions are given alone without any evidence paragraphs.
Note: For simplicity of training, train your model on 500 training instances and do parameter sweep on the same set of hyper-parameters given in HW6.

To use FAISS on Rockfish, please follow these instructions: `https://github.com/facebookresearch/faiss/blob/main/INSTALL.md`

Answer:
Link to your code on Github
Accuracy of model with gold paragraphs: ??
Accuracy of model without the gold paragraphs: ??
Briefly describe what do you take away from this comparison.

5 Prompting Language Models

1. Here we will prompt language models.[*] You should sign up for the OpenAI API, which lets you use GPT-3 a large, neural language model like the ones that we learned about in lecture.

   The OpenAI API is a paid service. OpenAI will give you a few dollars in credit when you first create your account. For this assignment, the cost should be less than that. For the first part of the assignment, we'll get warmed up by playing with the OpenAI API via its interactive Playground website. Later we'll see how to integrate it directly into our code.

   First, let's learn some basic terminology:

   - Prompt: the input to the model
   - Completion: what the model outputs

   Let's try it out. Then try pasting this prompt into the playground, pressing the "Generate" button, and see what it says:

   ```
   Daniel Khashabi's course on "self-supervised model" at Johns Hopkins is a great example of
   ```

   Your Response

   GPT-3 response here

   Now save its output for the end of the semester for your course reviews ... just kidding!!

2. There are several controls on the right hand side of the playground. These are:

   - **Engine:** GPT-3 comes in several different sized models. As the model sizes increase, so does their quality and their cost. They go in alphabetical order from smallest to largest. Here are several notable model sizes:
     - `Ada` - smallest, least costly model.
     - `Babbage`
     - `Curie`
     - `Davinci` - the largest and highest quality model until a while ago.

     You can read more documentation on these models here. There are newer versions such as GPT3.5 which have done through additional tuning with human feedback (we will discuss these later).
   - **Response length:** what's the maximum length (in tokens) that the model will output?
   - **Stop sequence:** you can specify what tokens should cause the model to stop generating. You can use the newline character, or any special sequence that you designate.

[*]Question credit: Chris Callison-Burch's AI course at UPenn.

- **Show probabilities:** allows you to highlight the tokens giving them different colors for how probable the models think they are.
- **Temperature and Top P sampling:** control how the model samples tokens from its distribution. Setting Temperature to 0 will cause the model to produce the highest probability output. Setting it closer to 1 will increase its propensity to create more diverse output.
- **Frequency Penalty and Presence Penalty:** two parameters that help to limit how much repetition there is in the model's output.

For the remaining parts of this section, set the engine to `davinci` so that it uses vanilla language model (no additional tuning with human feedback).

In-context learning: In addition to writing awesome reviews of your professors, you can design prompts to get GPT-3 to do all sorts of surprising things. For instance, GPT-3 can perform few-shot learning. Given a few examples of a task, it can learn a pattern very quickly and then be used for classification tasks. It often helps to tell the model what you want it to do.

Here's an example from the paper that introduced GPT-3. It shows a few-show learning example for correcting grammatically incorrect English sentences.

```
Poor English input:  I eated the purple berries.
Good English output:  I ate the purple berries.

Poor English input:  Thank you for picking me as your designer.  I'd appreciate it.
Good English output:  Thank you for choosing me as your designer.  I appreciate it.

Poor English input:  The mentioned changes have done.  or I did the alteration that you
requested.  or I changed things you wanted and did the modifications.
Good English output:  The requested changes have been made.  or I made the alteration that
you requested.  or I changed things you wanted and made the modifications.

Poor English input:  I'd be more than happy to work with you in another project.
```

What will you get?

Your Response

GPT-3 response here

3. **Instruction following:** In addition to few shot learning, GPT-3 and other large language models do a pretty remarkable job in "zero-shot" scenarios. You can give them instructions in natural language and they may produce remarkably good output.

   For example, let's try the following prompt:

   ```
   Correct this English text:  Today I have went to the store to to buys some many bottle of
   water.
   ```

   Show the output of at least 5 different engines to this prompt.

   Your Response

   Include the screenshots here and briefly explain why you think certain models are better at following instructions.

4. **Programmatic access:** You can use the playground to create code based on a prompt that you can then use in your Python projects. Click on the "View Code" button, and you'll get some code that you can convert into a Python function.

   Now let's use few-shot prompting to solve a few instance of BoolQ (HW6). Specifically, form a prompt by randomly selecting 8 demonstrations from BoolQ. Make sure to have a balanced prompt (50% 'yes's and the rest 'no's) and interleave them to prevent the recency and label-imbalance biases. Using this in-context demonstration, evaluate GPT3 `davinci` on a small subset of BoolQ instances (say, 30 examples).

Put your code on Github and share the link here, as well as the output:

Your Response

Answer: *Few-shot accuracy of GPT3-davinci on a random subset of test instances*
Link to your code

5. **Comparing to other models on HF:** Repeat the previous step with models from the BLOOMZ family hosted on Huggingface using programmatic calls. Since these models are free, evaluate them on a larger evaluation set (say, 100 instances). Note that unlike OpenAI models that are private, these models are open-source.

Your Response

Include the screenshots to model outputs and links to your code hosted on Github. Briefly describe how do the outputs of OpenAI's models compare against the open-source BLOOMZ models.

6. **Extra Credit:** Let's use a massive, web-scale language model! Specifically, let's use Petals which is BitTorrent-style distributed language model. Repeat previous step with Petals.

7. **Extra Credit:** Let's extend the previous step. Fine-tune Petals on 50 random instances from BoolQ and evaluate on the same set of instances from the previous step.

Optional Feedback

Have feedback for this assignment? Found something confusing? We'd love to hear from you!