Assignment 3

COMP3361: Natural Language Processing - University of Hong Kong

Spring 2024

1 Retrieval Augmented In-context Learning (25%)

In this assignment, you will explore advanced techniques in natural language processing, focusing on Retrieval Augmented In-context Learning (RAIL). This involves leveraging pretrained models and retrieval mechanisms to enhance the performance of machine learning models. Specifically, you will engage with contextual embeddings, similarity computation for retrieval, and augmenting few-shot learning with relevant examples.

1.1 Contextual Embedding

In this section, you will learn how to use a pretrained embedding model to obtain contextual embeddings. We will use scenarios from Assignment 1, Section 3, where we performed sentiment classification using GloVe word vectors. In this assignment, you will use the Instructor-base as our embedding model for the following questions.

- Encode the training and dev sentences we used in Assignment 1 Section 3.
- Use the same logistic regression method used in Assignment 1 to predict the sentiment labels of movie reviews. As before, only use the "train.txt" file to train the model. Tabulate and report the precision (P), recall (R), and F1 scores from the "dev.txt" file and compare this performance with the GloVe features.

1.2 Retrieval Relevant Examples

The embedding model serves not only to encode sentences into context embeddings for prediction tasks, but also to capture the text's semantic information, enabling similarity comparisons between documents.

Solving mathematical problems typically involves practicing with relevant examples before tackling new problems. Large language models operate similarly. In this section, we will assist the model in implementing a method to retrieve relevant examples from the GSM8K training set. This retrieval will be based on the similarity of these examples to the GSM8K test queries, using the embeddings we obtained.

- Embedding Generation: Use the Instructor model to encode both your query set (in this case, questions from the GSM8K test set) and your reference set (questions from the GSM8K training set).
- Similarity Computation: Calculate the cosine similarity scores between each query embedding and all reference embeddings.
- Example Retrieval: Retrieve the top 20 most similar examples from the reference set for each query. Save these examples in a JSON file for use in augmenting in-context learning prompts.

1.3 Generation with Huggingface Inference API

In Assignment 2, you utilized Huggingface Transformers to load the model. In this section, we will generate answers to GSM8K questions using a pre-trained language model via the Huggingface Inference API.

- Set up the Huggingface Inference API with codellama/CodeLlama-7b-hf and acquire an API token for authentication from `https://hf.co/settings/tokens`.
- Adapt your GSM8KCoTEvaluator using the provided API-based LLM class to execute 8-shot chain-of-thought inference. This should be done on the **first 30 examples in the test split**, similar to your approach in Assignment 2.

1.4 Impact of Quantity on Few-shot Prompting

This section investigates how the number of examples included in the few-shot prompt influences the quality of the generated answers.

- Modify the number of few-shot examples {1, 2, 4, 8} from the given GSM8K_EXAMPLES in your prompts. For each quantity of few-shot examples, generate answers for the **first 30 examples in the test split**. Evaluate the effects of the number of examples on model performance.

1.5 Retrieval Augmented Few-shot Prompting

Retrieval Augmented Few-shot Prompting is a comprehensive method that merges the principles of retrieval and few-shot learning. The fundamental idea is to enhance your prompts with the most relevant examples fetched from your dataset. This method is particularly useful when you are dealing with large datasets and need to optimize your model's learning process.

- For every individual query, you should load and utilize the top most similar examples from your training set. The number of examples should also vary within the set {1, 2, 4, 8}. Once you have these examples, arrange them in descending order of relevance. This order indicates that the most relevant example appears first, followed by less relevant ones.
- Compare the effects of retrieved examples to the settings with the same number of pre-defined examples.
- Order sensitivity is an important attribution of in-context learning. For the top 8 settings, rearrange the examples in ascending order of relevance. Evaluate the effects of this change.

Deliverables:

1. **Code (15%):** Implement code blocks.
2. **Write-up (10%):** Answer the following questions in your write-up:
   - **Q1.1:** Regarding the contextual embedding in Section 1.1, how does sentiment classification performance compare to your word2vec results in A1? Discuss potential reasons.
   - **Q1.2:** In Section 1.4, which discusses the impact of quantity, what trends do you notice when adding contextual examples? Do you think this trend will continue?
   - **Q1.3:** In Section 1.5, which covers retrieval-augmented in-context learning, how does this differ from Section 1.4? Analyze the reasons.
   - **Q1.4:** In Section 1.5, which arrangement yields better performance: in-context examples organized in descending or ascending order of relevance? Discuss the scenario.

2 Decoding Algorithms (25%)

In this exercise, you will implement a few basic decoding algorithms in the notebook: **greedy decoding**, **vanilla sampling**, **temperature sampling**, and **top-p sampling**.

2.1 Dataset

In this assignment, we focus on the open-ended story generation task (data available here). This dataset contains *prompts* for story generation, modified from the ROCStories dataset.

Fluency: The CoLA classifier is a RoBERTa-large classifier trained on the CoLA corpus (Warstadt et al., 2019), which contains sentences paired with grammatical acceptability judgments. We will use this model to evaluate fluency of generated sentences.

Diversity: The Count of Unique N-grams is used to measure the diversity of the generated sentences.

Naturalness: The Perplexity of generated sentences under the language model is used to measure the naturalness of language. You can directly use the perplexity function from HuggingFace evaluate-metric package for this assignment.

In the notebook, we have provided a wrapper function `decode()` that takes care of batching, controlling max length, and handling the EOS token. You will be asked to implement the core function of each method: **given the pre-softmax logits of the next token, decide what the next token is.**

2.2 Greedy Decoding

The idea of greedy decoding is simple: select the next token as the one that receives the highest probability. **Implement the `greedy()` function that processes tokens in batch.** Its input argument `next_token_logits` is a 2-D FloatTensor where the first dimension is batch size and the second dimension is the vocabulary size, and you should output `next_tokens` which is a 1-D LongTensor where the first dimension is the batch size.

The softmax function is monotonic—in the same vector of logits, if one logit is higher than the other, then the post-softmax probability corresponding to the former is higher than that corresponding to the latter. Therefore, for greedy decoding you won't need to actually compute the softmax.

2.3 Vanilla Sampling, Temperature Sampling

To get more diverse generations, you can randomly sample the next token from the distribution implied by the logits. This decoding is called sampling, or vanilla sampling (since we will see more variations of sampling). Formally, the probability of for each candidate token w is

$$p(w) = \frac{\exp z(w)}{\sum_{w' \in V} \exp z(w')}$$

where $z(w)$ is the logit for token w, and V is the vocabulary. This probability on all tokens can be derived at once by running the softmax function on vector $\mathbf{z}$.

Temperature sampling controls the randomness of generation by applying a temperature t when computing the probabilities. Formally,

$$p(w) = \frac{\exp\left(z(w)/t\right)}{\sum_{w' \in V} \exp\left(z(w')/t\right)}$$

where t is a hyper-parameter.

Implement the `sample()` and `temperature()` functions. When testing the code we will use $t = 0.8$, but your implementation should support arbitrary $t \in (0, \infty)$.

2.4 Top-p Sampling

Top-p sampling, or nucleus sampling, is a bit more complicated. It considers the smallest set of top candidate tokens such that their cumulative probability is greater than or equal to a threshold p, where $p \in [0, 1]$ is a hyper-parameter. In practice, you can keep picking candidate tokens in descending order of their probability, until the cumulative probability is greater than or equal to p (though there's more efficient implementations). You can view top-p sampling as a variation of top-k sampling, where the value of k varies case-by-case depending on what the distribution looks like. Similar to top-k sampling, the sampling probability among these picked candidate tokens should be proportional to their original probability implied by the logits, while summing up to 1 to form a valid distribution.

Implement the `topp()` function that achieves this goal. When testing the code we will use $p = 0.7$, but your implementation should support arbitrary $p \in [0, 1]$.

2.5 Evaluation

Run the evaluation cell. This will use the first 10 prompts of the test set, and generate 10 continuations for each prompt with each of the above decoding methods. Each decoding method will output its overall evaluation metrics: perplexity, fluency, and diversity.

Deliverables:

1. **Code (20%):** Implement code blocks.
2. **Write-up (15%):** Answer the following questions in your write-up:
    - **Q2.1:** In greedy decoding, what do you observe when generating 10 times from the test prompt?
    - **Q2.2:** In vanilla sampling, what do you observe when generating 10 times from the test prompt?
    - **Q2.3:** In temperature sampling, play around with the value of temperature t. Which value of t makes it equivalent to greedy decoding? Which value of t makes it equivalent to vanilla sampling?
    - **Q2.4:** In top-p sampling, play around with the value of p. Which value of p makes it equivalent to greedy decoding? Which value of p makes it equivalent to vanilla sampling?
    - **Q2.5:** Report the evaluation metrics (perplexity, fluency, diversity) of all 4 decoding methods. Which methods have the best and worst perplexity? Fluency? Diversity?

3 Written Problems (50%)

3.1 Multi-Choice (20%)

1. Select the answer that includes all the skip-gram (word, context) training pairs for the sentence the cat ran away, for a window size k = 2 from target.

    A) [the, cat], [the, ran], [cat, ran], [cat, away], [ran, away]

    B) [the], [cat], [ran], [away]

    C) [the, cat], [the, ran], [cat, the], [cat, ran], [cat, away], [ran, the], [ran, cat], [ran, away], [away, cat], [away, ran]

    D) [the, ran], [ran, cat], [cat, away]

E) [the, cat ran], [cat, the], [cat, ran away], [ran, the cat], [ran, away], [away, cat ran]

2. What if gradients become too large or small?

   A) If too large, the model will become difficult to converge

   B) If too small, the model can't capture long-term dependencies

   C) If too small, the model may capture a wrong recent dependency

   D) All of the above

3. Which of the following methods do not involve updating the model parameters? Select all that apply.

   A) Fine-tuning

   B) Transfer learning

   C) In-context learning

   D) Prompting

4. What range of values can cross entropy loss take?

   A) 0 to 1

   B) 0 to ∞

   C) -1 to 1

   D) $-\infty$ to 0

5. Can we use bidirectional RNNs in the following tasks? (1) text classification, (2) code generation, (3) text generation

   A) Yes, Yes, Yes

   B) Yes, No, No

   C) Yes, Yes, No

   D) No, Yes, No

6. Select the models that are capable of generating dynamic word embeddings, which can change depending on the surroundings of a word in a sentence.

   A) Bag of Words (BoW)

   B) Word2Vec

   C) GloVe

   D) T5

7. What makes Span Corruption a unique pretraining objective compared to traditional MLM?

   A) It involves masking and predicting individual tokens rather than spans of tokens.

   B) It only uses punctuation marks as indicators for span boundaries.

C) It masks contiguous spans of text and trains the model to predict the masked spans, encouraging understanding of longer context.

D) It requires the model to correct grammatical errors within the span.

8. In the transformer model architecture, positional encodings are added to the input embeddings to provide the model with information about the position of tokens in the sequence. Given a sequence length of L and a model dimension of D, which of the following PyTorch code snippets correctly implements the calculation of sinusoidal positional encodings?

A)
```
def positional_encoding(L, D):
    position = torch.arange(L).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, D, 2) * -(np.log(10000.0) / D))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position * div_term)
    return pe
```

B)
```
def positional_encoding(L, D):
    position = torch.arange(L).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, D, 2) * -(np.log(10000.0) / D))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position / div_term)
    pe[:, 1::2] = torch.cos(position / div_term)
    return pe
```

C)
```
def positional_encoding(L, D):
    position = torch.arange(L, dtype=torch.float).unsqueeze(1)
    div_term = 1 / (10000 ** (2 * torch.arange(D // 2) / D))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position * div_term)
    return pe
```

D)
```
def positional_encoding(L, D):
    position = torch.arange(L, dtype=torch.float).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, D, 2) * -(np.log(10000.0) / L))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position / div_term)
    return pe
```

9. In instruction tuning, what is the primary benefit of using natural language instructions for model fine-tuning?

A) Reduces the need for labeled data in supervised learning tasks.

B) Enables the model to perform zero-shot or few-shot learning on tasks it was not explicitly trained for.

C) Significantly decreases the computational resources needed for training large models.

D) Allows the model to improve its performance on specific tasks without fine-tuning.

10. How does Low-Rank Adaptation (LoRA) efficiently fine-tune large pre-trained models for specific tasks?

A) By freezing the original parameters and training a small set of new parameters introduced as adapters at certain layers, significantly reducing computational needs.

B) By applying low-rank matrices to adjust the attention weights, enabling task-specific tuning without extensively retraining the original parameters.

C) By pruning less important neurons based on initial assessments, simplifying the model for specific tasks with minimal performance impact.

D) By adding task-specific tokens to the model's vocabulary and fine-tuning their embeddings only, leveraging the existing model for seamless integration.

3.2 Short Answer (30%)

Question 3.1: A trigram language model is also often referred to as a second-order Markov language model. It has the following form:

$$P\left(X_1 = x_1, \ldots, X_n = x_n\right) = \prod_{i=1}^{n} P\left(X_i = x_i \mid X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}\right)$$

Question 3.1a: Could you briefly explain the advantages and disadvantages of a high-order Markov language model compared to the second-order one?

Question 3.1b: Could you give some examples in English where English grammar suggests that the second-order Markov assumption is clearly violated?

Question 3.2 We'd like to define a language model with $V = \{\text{the, a, dog}\}$, and $p(x_1 \ldots x_n) = \gamma \times 0.5^n$ for any $x_1 \ldots x_n$, such that $x_i \in V$ for $i = 1 \ldots (n-1)$, and $x_n = \text{STOP}$, where γ is some expression (which may be a function of n).

Which of the following definitions for γ give a valid language model? Please choose the answer and prove it.

(Hint: recall that $\sum_{n=1}^{\infty} 0.5^n = 1$)

1. $\gamma = 3^{n-1}$
2. $\gamma = 3^n$
3. $\gamma = 1$
4. $\gamma = \frac{1}{3^n}$
5. $\gamma = \frac{1}{3^{n-1}}$

Question 3.3 Given a small document corpus D consisting of two sentences: {"i hug pugs", "hugging pugs is fun"} and a desired vocabulary size of N=15, apply the Byte Pair Encoding (BPE) algorithm to tokenize the documents.

Assume the initial vocabulary includes individual characters and spaces as separate tokens. The BPE algorithm should merge the most frequent adjacent pairs of tokens iteratively until the vocabulary size reaches N=15.

Question 3.3a What is the final list of the desired vocabulary tokens?

Question 3.3b What is the final list of document tokens after reaching the desired vocabulary size?

Question 3.4 Let $\mathbf{Q} \in \mathbb{R}^{N \times d}$ denote a set of N query vectors, which attend to M key and value vectors, denoted by matrices $\mathbf{K} \in \mathbb{R}^{M \times d}$ and $\mathbf{V} \in \mathbb{R}^{M \times c}$ respectively. For a query vector at position n, the softmax attention function computes the following quantity:

$$\text{Attn}\left(\mathbf{q}_n, \mathbf{K}, \mathbf{V}\right) = \sum_{m=1}^{M} \frac{\exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right)}{\sum_{m'=1}^{M} \exp\left(\mathbf{q}_n^\top \mathbf{k}_{m'}\right)} \mathbf{v}_m^\top := \mathbf{V}^\top \text{softmax}\left(\mathbf{K}\mathbf{q}_n^\top\right)$$

which is an average of the set of value vectors $\mathbf{V}$ weighted by the normalized similarity between different queries and keys.

Please briefly explain what is the time and space complexity for the attention computation from query $\mathbf{Q}$ to $\mathbf{K}$, $\mathbf{V}$, using the big O notation.

4 Submission

Section 1&2 Coding Problems You should implement your solution with the provided ipynb template and include the written discussion in the text cells. Submit the **UniversityNumber.ipynb** to Moodle.

Section 3 Written Problems You should implement your answers with the LaTeXtemplate on Overleaf. Here is a tutorial on working with LaTeXand Overleaf. Submit the complied PDF **UniversityNumber.pdf** to Moodle.