Advanced Natural Language Processing - Exercise 1

Malak Laham

Github repository: https://github.com/malak-codes/ANLP.git

Theoretical Part

Q1. **Three QA Datasets That Annotate Intrinsic Language Understanding Concepts**

1. **SQuAD (Stanford Question Answering Dataset):** questions posed by crowdworkers on a set of Wikipedia articles where the answer to every question is a segment of text, or span, from the corresponding reading passage. It tests an intrinsic task since it is sort of a reading comprehension dataset where in order for the model to answer the question, it has to understand the language, comprehend contextual meaning, interpret sentence-level semantics, and locate relevant information within a text.
2. **SWAG / HellaSwag:** given a sentence, the task for the model is to choose the most likely continuation for it. Thus, the task requires the model's understanding of event causality, temporal order, and semantic compatibility making a dataset which tests the intrinsic concepts of language understanding.
3. **BoolQ (Boolean Questions):** yes/no questions paired with short paragraphs, requiring models to infer the correct answer.
   This dataset tests intrinsic understanding by evaluating how well a model can reason over sentence meaning, interpret negation, and make semantic inferences based only on textual content.

Q2. (a)

1. Self-Consistency

Brief Description:

Instead of using a single output from the model, self-consistency samples multiple reasoning paths (different chains-of-thought) and chooses the final answer by majority voting.

Advantages:

Improves reliability and accuracy by allowing the model to solve problems in multiple ways. Then given that in general it is a good model, choosing based on the majority vote lowers the chances of flawed or incorrect reasoning. So it allows the model to "think" in different directions before settling on an answer, ultimately leading to more dependable performance on complex and reasoning-intensive tasks.

Computational Bottlenecks

This method requires the model to generate many different reasoning paths, and each sample involves a full forward pass through the model - from the prompt to the final output. Thus, to give the final response, one must wait for all reasoning paths to complete in order to do majority voting.

Per research and as expected, this method also requires heavy GPU usage as described below:

-Each forward pass consumes GPU memory for the input, intermediate activations, and output tokens.

-Running many passes sequentially stretches GPU resources, especially on long prompts or large models.

-This can exhaust memory or reduce batch sizes, particularly problematic in resource-limited environments.

Parallelization

Per research, self-consistency is highly parallelizable, making it scalable if hardware resources allow. This is because each reasoning path is independent, and paths don't need to share intermediate results or wait on each other.
With GPUs that allow it, we can run multiple forward passes simultaneously:

2. Verifiers

Brief Description:
A method where multiple reasoning chains are generated, and a separate model or scoring mechanism is used to evaluate and select the most accurate, consistent, or plausible response. This can involve verifying the full answer of a single chain-of-thought or scoring its intermediate steps and continuing only the most promising reasoning paths.

Advantages:

By verifying the CoT, this method filters out incorrect or illogical responses and enables modular and flexible pipelines. This can highly increase accuracy without changing the model itself.
Computational Bottlenecks

Time usage: For each generated answer or reasoning chain, the verifier must evaluate its quality, which usually means running a separate model (or the same model in a different mode). Thus, if we generate N candidate answers and verify all of them, we double the inference time: one pass to generate each candidate and another one to verify it. In addition, verification could involve the evaluation of different aspects such as logical consistency, factuality, and more aspects further increasing the verification and inference time.
After verification and evaluation is done, the CoTs and answers are reranked and resorted which also requires further computation and time.

Memory Usage: Both the generated candidates and their verification scores need to be stored temporarily.

Parallelization

Since each candidate output is verified independently from the others, then all verifier evaluations can be run in parallel, whether on multiple GPU cores, multiple devices, or a single device with batched inputs.
Pipelining to parallelization: since generation and verification are separate phases, they can be pipelined: while some outputs are being verified, others can be generated.

3.Planning

Planning involves the model having to first generate a plan which could include a set of steps, intermediate goals, or an outline before producing the final answer. This plan is then used to guide the final response generation.

Advantages

Since planning encourages the model to explicitly think ahead, it helps the model structure its approach, which could be very useful in complex tasks. Such an approach inherently improves the consistency of the model since the plan serves as some kind of "sanity check" for the model and at the same time acts as a "verifier" since the model is not only generating an answer but also a specific plan to follow in order to finally reach an answer. Such a method helps the model avoid drifting into unrelated or incoherent answers especially in long and complex tasks.

In another sense, for the model's developers, this makes debugging easier if the output is wrong (the reason for the fault could be traced to either the plan or solely the final answer.)

Computational Bottlenecks

-Since the model must generate a plan and then the actual answer, this extra step adds an extra forward pass which kind of "doubles" the total inference time.

-If planning is done with a separate prompt or model, this would require careful coordination / pipelining / tokenization between the planning stage output and the input for generating the final answer.

- If plans are long, feeding the plan back into the model may push us close to the model's context length limit.

Parallelization

Not possible: Within a single example, planning must come before the answer. Thus, we cannot parallelize the two phases for the same input.

4. Backtracking

Brief Description:
If a model detects a mistake in reasoning, it "goes back" to a previous step and tries a different reasoning path.

Advantages:

Instead of canceling out an entire CoT, and starting a new one from scratch, upon detection of a mistake in reasoning, the model backtracks to the mistake and carries on from then on. This increases correctness by recovering from early mistakes.

Reusing previously correct reasoning steps can be computationally cheaper than generating multiple complete CoTs.

Computational Bottlenecks:

Time Complexity

- Each backtrack adds another forward pass; thus if the model backtracks multiple times or tries several alternatives for each branch, this increases computation time for generating a single answer.

- Tracking states, identifying error points, and deciding where and how to branch introduces complexity and non-linearity into the inference pipeline.

Memory Complexity

-Must keep track of partial CoTs or step-level states, which increases memory requirements during generation.

Parallelization:
Partially. Alternative paths can be explored in parallel, but the process has sequential dependencies so it's not fully parallelizable.

5. Self-Evaluation

Brief Description:
The model critiques or scores its own answer, often using prompts that ask it to reflect on correctness or confidence. We note, that as said in the lecture, it is not clear exactly how the evaluation is done since companies are discrete on their models, but we can expect it to include verification on different parts of the generated answer with respect to the prompt and facts.

Advantages:

-Improves output quality by filtering or reranking
-Encourages more thoughtful generation and higher accuracy.
-Doesn't require planning or internal state tracking.

Computational Bottlenecks

-Each self-evaluation requires additional prompts or tokens, increasing the number of inference steps and computation time.

Parallelization

In the examples we saw on self-evaluation in the lecture, it seemed as if the model was asking itself questions when considering different aspects of the prompt it is asked. Which means it is similar to backtracking in this sense. If the evaluation is done in this manner, where at some points the model diverts into multiple CoTs, then the generation can be parallelizable from these diversion points and on; however, if the model evaluates itself in the end on a single CoT, then it isn't parallelizable as we must wait for the model to complete a single answer and only then evaluate.

6. Using Multiple Small Language Models and Voting By Majority

Description:

As we saw in class, for some tasks, smaller language models outperform larger language models. So, if we use majority voting on multiple language models during inference time, this increases the inference time scaling and we expect to see better results.

Advantages

Using multiple language models, each of which could be fine-tuned quite differently can serve as multiple perspectives/ different attention views on the problem and thus when taking the majority voting from these smaller models, our answer is likely taking into consideration several aspects and increasing the overall accuracy of it.

Smaller models can be run in parallel to receive an answer from each of them and then do a majority voting thus computation time isn't necessarily different or larger.

Computational Bottlenecks

Memory usage: keeping track of multiple models and their outputs simultaneously increases memory usage.

Parallelization

We can run the smaller models simultaneously since they are independent of each other and do majority voting.

(b) Suppose you must solve a complex scientific task requiring reasoning, and you have access to a single GPU with large memory capacity. Which method would you choose, and why?

Given a complex scientific task requiring reasoning, the most critical need is reliable, structured, and multi-step thinking. Self-Consistency addresses this by sampling multiple reasoning paths (different chains-of-thought) and selecting the final answer through majority voting. This approach filters out accidental or flawed reasoning and favors frequently recurring, well-structured outputs, increasing the likelihood of capturing correct logic. Although the method's main computational bottleneck lies in the need to run multiple forward passes, a single GPU with large memory capacity can efficiently handle this through batching and parallel generation of reasoning paths. Importantly, Self-Consistency is particularly well-suited for single-GPU setups because it uses only the base model - no additional models, verifiers, or orchestration logic are required. This makes it memory-efficient and simple to deploy, with each reasoning path being independent and stateless. In contrast, methods like Verifiers and Backtracking require additional components or introduce complex control flow, making them harder to manage within the constraints of a single device. Overall, Self-Consistency offers a clean, effective, and scalable solution for high-quality reasoning on a single high-memory GPU.

Practical Part 2.1.2

• Did the configuration that achieved the best validation accuracy also achieve the best test accuracy?

The configuration that achieved the best validation accuracy (0.8505) is:

lr=2e-05, batch size=16, epochs=5.

Its test accuracy is: 0.8510
So **yes** it achieved the highest accuracy both on the validation and test set.

• Qualitative analysis: Compare the best and worst performing configurations. Examine validation examples where the best configuration succeeded but the worst failed. Can you characterize the types of examples that were harder for the lower-performing model?

The worst-performing configuration (lr=0.001, bs=32, ep=1) consistently predicted all sentence pairs as paraphrases (label 1), resulting in zero true negatives and a large number of false positives. This behavior likely stems from its high learning rate, larger batch size, and insufficient training time (1 epoch), which caused the model to decide on a simplistic decision strategy which was to label all sentences as paraphrases. Per research, it turns out that the MRPC dataset has 67.6% paraphrases in the training set; thus, we assume that the weak model configuration was biased based on the majority of the training examples and matches our prediction for its decision strategy. In contrast, the best-performing model (lr=2e-5, bs=16, ep=5) had enough training (5 epochs) and smaller batch size, allowing it to learn finer semantic distinctions and having no such extreme bias and reaching a higher accuracy.