Advanced NLP course – EX1

Open questions and programming exercise analysis

Dorin Shteyman (206721102)

Github link: https://github.com/dorin133/ANLP-EX1.git

2. After performing hyperparameter search, use all trained checkpoints to make predictions on the test set. Then, answer the following:
   - Did the configuration that achieved the best validation accuracy also achieve the best test accuracy?
   - **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?

Qualitative analysis:

The tested configurations and their evaluation + test accuracy:

- **WORST -** epoch_num: 3, lr: 3e-05, batch_size: 16, eval_acc: 0.799, test_acc: 0.7820
- **BEST -** epoch_num: 2, lr: 0.0001, batch_size: 8, eval_acc: **0.863**, test_acc: **0.8157**
- epoch_num: 1, lr: 0.0001, batch_size: 4, eval_acc: 0.826, test_acc: 0.7919
- **WORST -** epoch_num: 5, lr: 2e-05, batch_size: 32, eval_acc: 0.804, test_acc: 0.7786

As can be observed, **the configuration with the best validation accuracy also achieved the best test accuracy**, and they are not so far from each other. This indicates the model's configurations are good and promote generalization while avoiding overfitting.

Comparison of the best and worst performing configurations: many samples the worst configuration fails to recognize **as equivalent** (i.e., labelling them 0 when the true label is 1) are when both sentences share a similar prefix of 3-10 words. This overlap ironically causes the worst model to mistakenly identify them as carrying different meaning, probably because there are more samples with the same prefix that are not equivalent, making this model biased to predict them as not equivalent for this sentences' structure. Another reason for that could be worst semantic/entity linking of similar expressions (e.g., turned from sour to sweet / have changed their minds) of the same entity or it is appearing at a different location in the sentence (e.g., busing/school busing).

Example 1:

> Sentence 1: The decision came a year after Whipple ended federal oversight of the district 's racial balance, facilities, budget, and **busing**.
>
> Sentence 2: The decision came a year after Whipple ended federal oversight of **school busing** as well as the district 's racial balance, facilities and budget.
>
> (True) Label: 1, (Worst model) Label: 0

Example 2:

> Sentence 1: Federal regulators have **turned from sour to sweet** on a proposed $ 2.8 billion merger of ice cream giants Nestle Holdings Inc. and Dreyer 's Grand Ice Cream Inc.
>
> Sentence 2: Federal regulators **have changed their minds** on a proposed $ 2.8 billion merger of ice cream giants Nestle Holdings and Dreyer 's Grand Ice Cream.

(True) Label: 1, (Worst model) Label: 0

Question 1:

1. Question answering (QA) can be an expressive format for annotating both intrinsic as well as extrinsic tasks. **List three QA datasets that use QA to annotate intrinsic concepts. For each, write a short explanation (1-2 sentences) for why it measures an intrinsic property of language understanding.**

SQuAD (Stanford Question Answering Dataset):

Consists of questions on a set of Wikipedia articles. It is designed to evaluate a language model's ability to answer factual questions from a given passage. To perform well on this dataset, the model must use its following intrinsic concepts: lexical and syntactic understanding (recognize paraphrases and varying word orders), semantic understanding and contextual awareness (understanding how different sentences semantically relate to identify the exact areas of text in the passage that answers the question).

TriviaQA:

Containing question-answer pairs of trivia. This dataset evaluates LLM's intrinsic concepts such as world knowledge (model's recall of dates, events, and factual other information), entity resolution (resolving entity references within a passage like "she", "it"). In general, TriviaQA performance measures how well an LLM can understand handle open-domain knowledge and retrieve correct answers.

Gsm8k:

Contains multi-step arithmetic problems (in words). It is designed to assess the intrinsic concepts of **mathematical reasoning** of language models. This is because to solve GSM8K problems LLMs must use reasoning skills and can't base on memorized arithmetic, because they have never seen these problems before. Other two important intrinsic concepts this dataset requires the model to have are problem decomposition (decomposing the problem to individual subproblem parts to perform the right calculation flow for the final solution), semantic understanding of mathematic concepts (translating natural language to correct mathematical representation entities to construct equations).

Questions 2:

2. In class we discussed several methods to implement inference-time scaling.
   (a) For each method we covered, answer the following:
   - Provide a brief description of the method.
   - Outline its advantages.
   - Identify its computational bottlenecks (i.e., the resources heavily consumed during its execution).
   - Indicate whether the method can be parallelized.

   (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?

(a) In class, we focused on Inference-time scaling methods for increasing accuracy, *not* on methods who alleviate computing and memory requirements (quantization, distillation, etc.):

CoT/Self consistency:

Description: CoT prompting (like we utilize in our final project of the course ☺) guides the model to generate intermediate reasoning steps before the final answer, to allow for step-by-step problem-solving. Self consistency is similar, only generating N CoT outputs and choosing the final answer to be the one most consistent reasoning path w.r.t other outputs. Consistency can be determined by verifiers/sentence similarity/final output.

Advantages: CoT prompting makes model decisions more interpretable and traceable. In addition, it was empirically shown [1] to increase accuracy on complex reasoning tasks who require multi-step composition of sub-problems. Moreover, further accuracy gain was empirically shown to be obtained with choosing the answer with most consistent CoT path out of N possible reasoning paths generated [2]. Another advantage of CoT is obtaining each full reasoning path within a single inference of the model (no reliance on verifiers/additional runs of assistant models/ etc.).

Computational Bottlenecks: increases inference time by a large factor (usually) compared to only generating the final answer. This also increases the token generation cost (more tokens to attend to means more compute cost and higher memory cost to save the full KV cache). For self-consistency, the multiple sampled paths can further increase the inference cost (especially if N>batch-size and we can't produce all samples within a single batch on our hardware). Depends on the complexity of the consistency metrics, its calculation can potentially add further noticeable overhead.

Parallelizability: each CoT sample is sequential due to dependency between steps, so a single CoT can't be parallelized. However, multiple CoT reasoning paths (for self-consistency) can be batched and generated (and compared) in parallel.

Rejection sampling/Best-of-N Sampling (usually applied by using an inferior model like in Hassid's paper [3]):

Description: generate multiple candidate outputs N and select the best one based on a scoring function.

Advantages: substantial improvement in robustness, by reducing hallucinations or logical errors. In some settings, running multiple times an inferior model like in [3] can outperform a single run of a superior and larger model, which enables high accuracy on low-resource hardware that can't fit a large model. Compared to Self-consistency (CoT generations), answers can be shorter and still be

produced when the context window is small (especially relevant for small models who were pretrained using small context length)

Computational Bottlenecks: the selection of the final output requires scoring calculations, which can introduce additional overhead. Also, inference time can scale up to a factor of N (i.e., the number of samples) in a scenario of batch size=1 serving (like on low bandwidth PC/mobile hardware).

Parallelizability: the N generations can be parallelized and be completed in $\left\lceil \frac{N}{BatchSize} \right\rceil$ generation steps. The final selection step may or may not be sequential, depending on scoring method. For instance, a scoring function of each generated outputs' likelihood can be parallelized because the score depends only on the content of each answer for itself.

Verifiers:

Description: Use a secondary model or modules (e.g., regular expressions for math or unit-tests for code) to evaluate the correctness of each candidate output or reasoning step.

Advantages: Improves factual accuracy and can be more reliable than scoring functions which are usually calculated by LLM probability assignments to tokens (can be corrupted due to LLM's bias, hallucinations, etc.). Can also be applied on mid-steps and accept/reject reasoning paths in their early stages, which can save compute and promote accuracy by outputting answers who passed all mid-step verifications. In cases like math or code, verifiers are lightweight and fast compared to a strong LLM with high parameter count.

Computational Bottlenecks: Sometimes verifiers are LLMs themselves and produce substantial overhead. In addition, if we need to run multiple verifiers for each solution step (like in CoT), this overhead can occur sequentially if previous verifiers' (on previous solution steps) outputs and runs depend on the output of the current verifier. Another case is when multiple verifiers evaluate the solution, where overhead is induces by them all.

Parallelizability: Verifiers are parallelizable across candidates/verification stages, but sequential if used step-by-step + depend on each other's previous outputs.

(b) Large-memory GPU allows running long outputs and multiple samples (batch) in parallel. Meaning, we can run both **CoT and Best-of-N based on score functions / unit tests / verifiers** (depending on the use case, and the availability of strong verifiers or good unit tests for code). To optimize efficiency, we'll prefer to set N the sample size to be the batch size, to avoid using computation time past the longest generated CoT sequence.

Since the task is defined as complex and requires reasoning, CoT with long context (test-time compute scale like in o1 [4]) is the right approach for each of the N samples (fitted in a single batch). This way we best leverage both the compute and memory of the GPU.

Side note – from personal experience, it's worth to first estimate the generation complexity (a.k.a the required average number of steps to generate a *single* correct answer given the maximal sequence length the LLM allows) and based on this to build to the optimal scheme for the current scenario.