Reading notes : Assignment 1

**Abstract:**

* LMs are prone to hallucinations, and struggle with arithmetic and factual information.
* To solve this, they could rely on external tools and apis that would give them access to information and tools that may reduce their loss, making them more accurate.
* This paper uses apis to a calculator, Q&A system, a search engine, a translation system, and a calendar
* Toolformer is based on GPT-J

**Intro:**

* LMs have limitations regardless of their excellent performance. They don’t have access to real time data and more recent events, such as chatgpt which only had information till 2021. They also have difficulty understanding languages with less number of documents and coverage(Lin et al 2021) and mathematical skills.
* A lot of these issues could also be seen in chatgpt implementations, where all of these problems still exist however OpenAI is solving them through finetuning their model.
* Giving them external tools such as search engine access, calculators, and calendars should help them overcome a lot of these obstacles.
* These models should be done without human annotations, and the model should figure out without major human interactions when it should use the APIs
* They should also not lose any of their generality.

**Procedure:**

* Create a handful of samples that involve API usage. Create entire datasets from scratch (schick and schutze 2021b) using in context learning(Brown et al 2020). Use a self-supervised loss to determine whether the API calls decrease the loss or not.
* The model should not lose generality, and this is done by letting the model decide when to use the tool.
* Use the same dataset that it was trained on to preserve generality and language modelling task performance

**Approach->**

* Use another token, which is the api token. This consists of a tuple, (ac, ic); the first is the name of the API, and the second is the input of the API.

**Representation of API calls:**

* Each API call is represented as a tuple c = (ac, ic), where ac is the name of the API and ic is the corresponding input.
* The linearized sequences of the API call, not including and including its result, are denoted as follows:
  + e(c) = **<API> ac(ic) </API>**
  + e(c, r) = **<API> ac(ic) → r </API>**, where "→" indicates the result.

**Augmentation of dataset with API calls:**

* Given a dataset C = {x1, ..., x|C|} of plain texts, it is converted into a dataset C\* augmented with API calls in three steps:
  1. Exploiting in-context learning ability to sample potential API calls.
  2. Executing these API calls.
  3. Filtering obtained responses to predict future tokens.
* After filtering, API calls for different tools are merged to create the augmented dataset C\*.

**Sampling API Calls:**

* For each API, a prompt P(x) is created to encourage the LM to annotate examples with API calls.
* Candidate positions for API calls are sampled based on the probability assigned by the LM to start an API call.
* A threshold τs is used to select the top positions, and if more than k positions are selected, only the top k are kept.
* API calls are sampled given the prefix sequence and end-of-sequence token, discarding examples where the end token is not generated.

**Executing API Calls:**

* All generated API calls are executed to obtain corresponding results, the method of execution depending on the API itself.
* Execution methods can include calling another neural network, running a Python script, or using a retrieval system.

**Filtering API Calls:**

* API calls are filtered based on their impact on model prediction.
* Weighted cross-entropy loss is calculated for the model with and without the API call and its result as a prefix.
* API calls are kept if adding them and their result reduces the loss by at least a threshold τf compared to not using them.

**Model Finetuning:**

After the filtering is done for the API calls, the API calls are then carried out and they are interleaved with the original input.

* **Model Finetuning After Sampling and Filtering:**
  + Process involves merging remaining API calls with original inputs.
  + Construct new sequence x\* by interleaving API calls and original text.
  + For input text x = x1, ..., xn with corresponding API call and result (ci, ri) at position i:
    - Construct x\* = x1:i-1, e(ci, ri), xi:n.
    - Repeat process for texts with multiple API calls.
  + Resulting dataset C\* is augmented with API calls.
  + Finetune model M on C\* using standard language modeling objective.
  + C\* contains same texts as original dataset C, with inserted API calls.
  + Finetuning on C\* exposes model to same content as finetuning on C.
  + API calls inserted where they help model predict future tokens.
  + Finetuning on C\* enables model to decide when and how to use tools based on its own feedback.

How are the API calls processed?-> While decoding, when the “->” token is created, the decoding process is interrupted and the api call is carried out and the response is replaced.

Tools Used:

* **Tool Descriptions:**
  + **Question Answering:**
    - Uses Atlas, a retrieval-augmented LM, for answering factoid questions.
  + **Calculator:**
    - Performs basic arithmetic operations, rounding results to two decimal places.
  + **Wikipedia Search:**
    - Returns short text snippets from Wikipedia based on search term.
    - Utilizes BM25 retriever for indexing Wikipedia dump from KILT.
  + **Machine Translation System:**
    - Translates phrases from any language into English.
    - Utilizes 600M parameter NLLB model for multilingual translation.
  + **Calendar:**
    - Returns current date without input.
    - Provides temporal context for predictions requiring time awareness.

**Experimentation**

**Dataset Generation:**

Throughout all of our experiments, we use a subset of CCNet (Wenzek et al., 2020) as our language modeling dataset C and GPTJ (Wang and Komatsuzaki, 2021) as our language model M.

* Employ GPTJ as language model M throughout experiments.
* Define heuristics for selecting texts in C where API calls are likely to be beneficial.
* Detailed heuristics provided in Appendix A.
* Generate augmented dataset C\* from C, following steps in Section 2.
* Filter out examples where all API calls were eliminated.
* Apply weighting function to ensure API calls occur close to relevant information.
* Individual thresholds (τs and τf) chosen for each tool to ensure sufficient examples.
* Relevant statistics of final dataset with API calls shown in Table 2.
* **Model Finetuning:**
  + Finetune model M on C\* with batch size of 128 and learning rate of 1e-5.
  + Linear warmup for first 10% of training.
  + Detailed finetuning procedure in Appendix B.
* **Baseline Models:**
  + Compare various models:
    - GPT-J: Regular GPT-J model without finetuning.
    - GPT-J + CC: GPT-J finetuned on subset of CCNet without API calls.
    - Toolformer: GPT-J finetuned on augmented dataset C\* with API calls.
    - Toolformer (disabled): Same as Toolformer, but API calls disabled during decoding.
  + Additionally compare to larger models OPT (66B) and GPT-36 (175B).

This is different from their previous work on tool usage (Gao et al 2022) where models are provided with “dataset-specific examples of how the tools can be used to solve a task.

Standard Greedy decoding is used for toolformer(cite lecture). Use APIs when it is one of the k most likely tokens. At most one API call per input to prevent infinite loops.

**LAMA Evaluation:**

* + Evaluate models on subsets of LAMA benchmark: SQuAD, GoogleRE, and T-REx.
  + Task involves completing short statements with missing facts.
  + Filter out examples where mask token is not final token for left-to-right processing.
  + Use lenient evaluation criterion checking if correct word is within first five predicted words.
  + Prevent Toolformer from using Wikipedia Search API to avoid unfair advantage.

**Results:**

* + All GPT-J models without tool use show similar performance.
  + Toolformer outperforms baseline models significantly, improving by 11.7, 5.2, and 18.6 points respectively.
  + Outperforms larger models OPT (66B) and GPT-3 (175B) despite their size advantage.
  + Toolformer primarily relies on question answering tool (98.1%), occasionally using different tools (0.7%), or none (1.2%).

Top of Form

Bottom of Form

Math Datasets:

* As required output is always a number, we take y\_hat as the first number predicted by the model. Toolformer achieved better performance even when the API calls are disabled, however this could be attributed to the fact that they were finetuned on many examples of API calls.

Question Answering

* Web Questions (Berant et al., 2013), Natural Questions (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017). These were the datasets considered by Brown et al.
* Evaluation done by checking is the correct answer if produced in the first 20 words predicted by the model.
* There's a concern that enabling the question answering tool for Toolformer would make solving the tasks too easy or trivial, especially since the underlying question answering system was fine-tuned on Natural Questions, implying it's already specialized for this task.
* Mostly relies on Wikipedia API to find relevant info, however lags behind GPT3 model which has a larger amount of params.

Multilingual Question answering

* Context paragraph is in English, and the question is some other language. It needs to understand both the paragraph and the question to answer the questions. The idea here is that the tool should benefit by doing a translation (using the translation API tool).
* Evaluation done on percentage of times the model generated answer contains the answer(capped at 10 words)
* Results: Finds that API tools enable toolsformer to improve performance, however, for hindi it is not that effective and barely used compared to other languages. Doesn’t always perform better than GPT-J as for some languages, finetuning on CCNet deteriorates performance.
* OPT and GPT-3 perform surprisingly week on all languages, however this could be attributed to it giving answers in other languages.
* OPT and GPT-3 perform poorly across all languages, especially in providing answers in English as instructed.
* GPT-J may avoid this issue due to its training on a more extensive multilingual dataset, including the EuroParl corpus.
* GPT-J and GPT-3 are evaluated on a variant of MLQA with both context and questions in English.
* In this setup, GPT-3 outperforms all other models, suggesting that its weak performance on MLQA might have specific reasons.

Temporal datasets:

* This is used to investigate it’s performance when using the calendar API. All models are evaluated on TEMPLAMA(Dhingra et al 2022) and dataset: DATESET.
* This involves more information on recent events, and knowing the current date etc.
* Toolformer achieves better results compared to all the other models. However, the majority of calls were made to the Wikipedia API. This makes sense given that entities in TEMPLAMA are often so specific and rare that knowing the exact date would be of little help.
* This is not the best way to evaluate, as the calendar tool could be used instead to fetch the current date, and then use another API (which is not done here as it only supports 1 API call per input). Also all API calls are done and sampled independently.
* However for DATESET, it is used the calendar tool for 54.8% of the examples

Language Modelling

* Testing done to ensure that toolformer doesn’t degrade through finetuning of the model. Thus we evaluate on two language modeling datasets: WikiText and 10,000 random documents selected from CCNet.

**Scaling Laws:**

* Investigate effect of external tool usage on LM performance with varying model sizes (GPT-J and GPT-2 family).
* Applied approach to smaller GPT-2 models with parameters: 124M, 355M, 775M, and 1.6B.
* Utilized three tools: question answering system, calculator, and Wikipedia search engine.
* Ability to leverage tools only significant around 775M parameters.
* Gap between predictions with and without API calls remains high even for larger models.

**Analysis:**

* Modified decoding strategy introduced to generate "<API>" token if it's among the k most likely tokens.
* Performance on T-REx subset of LAMA and WebQS increases with increasing k.
* Model calibrated to some extent for k = 1, making API calls for examples where performance would be particularly poor without API calls.
* Qualitative analysis of API calls generated with approach for different APIs, considering usefulness in context.

**Related Work:**

* Various approaches augment LMs with additional textual information during pretraining, including metadata, HTML tags, Wikipedia markup, or related texts from information retrieval systems.
* Toolformer differs by learning to explicitly ask for the right information without additional textual information.
* Approaches equip LMs with ability to use external tools such as search engines, web browsers, calculators, translation systems, and Python interpreters.
* Toolformer learns to use tools without specific prompts, unlike other approaches requiring task-specific examples.
* TALM explores similar self-supervised objective for teaching model to use calculator and search engine, but in settings where model is fine-tuned for downstream tasks.

**Limitations:**

* Toolformer unable to use tools in chain or interactively.
* Sensitivity to exact wording of input when deciding API calls.
* Sample-inefficient for certain tools, requiring iterative application.
* Currently, does not consider computational cost of making API calls.

**Conclusion:**

* Introduced Toolformer, self-supervised LM learning to use tools via API calls.
* Finetuned on sampled API calls, filtered based on reducing perplexity on future tokens.
* Improves zero-shot performance of GPT-J model, outperforming larger GPT-3 model on various downstream tasks.