Week 5 Thursday Paper 1: [Toolformer: Language Models Can Teach Themselves to Use Tools]

Explained by

How does AI learn to use tools? - Toolformer explained

Supplemental:

LangChain Explained in 13 Minutes | QuickStart Tutorial for …

[What is LangChain? | IBM]

1. What is Toolformer and what are its applications?

Ans: Toolformer is a language model designed to autonomously use external tools via simple APIs, combining the broad capabilities of large language models with the precision of specialized tools. It is trained to decide which APIs to call, when to call them, what arguments to pass, and how to best incorporate the results into future token prediction, all in a self-supervised manner requiring only a few demonstrations for each API. The model incorporates a variety of tools, including calculators, Q&A systems, search engines, translation systems, and calendars. This approach allows Toolformer to achieve significantly improved zero-shot performance across a range of downstream tasks, often competing with much larger models while maintaining its core language modeling abilities.

2. What special tokens does it use? Why are they important?

Ans: Toolformer uses special tokens to mark the start and end of each API call within a text sequence. These tokens are “<API>” and “</API>” for marking the beginning and end of an API call, and “→” for separating the API call from its result. In practice, these are represented by the token sequences “ [”, “] ”, and "->" to work without modifying the existing language model's vocabulary. These special tokens are crucial as they allow the seamless insertion of API calls into text, ensuring that inputs and outputs for each API can be represented as text sequences. This method supports the model's ability to use different tools through API calls in a structured and recognizable format, facilitating the integration of external data or computations directly into the text flow.

3. How are the responses of the API incorporated into the LLM (or it’s context window)?

Ans: The responses from the API are incorporated into the LLM during both the finetuning and inference stages:

Fine tuning: After sampling and filtering API calls, Toolformer integrates these calls and their responses with the original input texts. This integration forms a new sequence, where the API call and its response are inserted before the point they are needed in the text. This approach prevents disruption of the text's natural flow and maintains alignment with patterns seen during pre-training, avoiding negative impacts on perplexity. This new dataset, augmented with API calls, is used to finetune the model using standard language modeling objectives. Crucially, apart from the inserted API calls, the augmented dataset remains identical to the original dataset, enabling the model to learn from content it was initially trained on while incorporating external tool usage.

Inference: During text generation, the model performs regular decoding until it produces the "→" token, indicating it expects an API call response. At this point, the decoding process is interrupted to call the appropriate API and obtain a response. The decoding then resumes, incorporating both the response and the closing `</API>` token into the generated text. This mechanism enables the model to seamlessly integrate external information into its output in real-time.

4. Why is filtering of API calls done? What loss function is used and why?
Ans: Filtering of API calls is performed to ensure that only useful API calls are retained, which helps in reducing unnecessary or irrelevant external interactions. This is crucial for maintaining efficient and pertinent API usage, especially given the potential costs and constraints associated with API calls.

The loss function used for filtering is based on a weighted cross-entropy loss, compared in two different scenarios:

Li+ is the loss when the API call and its result are provided to the model as a prefix. This represents the scenario where the model has access to the output of the API call, potentially aiding in future token prediction.

Li− is the minimum loss obtained either by doing no API call at all or by doing an API call but not providing the response. This represents scenarios where the model either does not use the API at all or uses the API but does not receive any helpful information from it.

The filtering criterion is that an API call is kept if (Li−) − (Li+) is greater than or equal to a threshold τf. This means that the API call is considered useful and retained if adding the call and its result to the model's input reduces the loss by at least τf compared to not making the call or making the call but not obtaining any result. This approach ensures that the model only retains API calls that contribute positively to its ability to predict future tokens, thereby enhancing its efficiency and effectiveness.

5. How is the model fine-tuned?
Ans: The original text dataset is converted into an augmented dataset which includes API calls. This augmentation involves sampling a large number of potential API calls based on the model's in-context learning capabilities, executing these calls to obtain responses, and applying a filtering criterion to ensure only beneficial responses are retained. The augmented dataset incorporates API calls that help the model in predicting future tokens, enhancing its utility and relevance.

Filtering is critical to ensure only useful API calls are included. This involves comparing the weighted loss of sequences with and without API calls and their results. An API call is retained if it significantly improves the model's ability to predict subsequent tokens, determined by a predefined threshold.

The final step involves fine-tuning the language model on the augmented dataset. This fine-tuning allows the model to adapt to the inclusion of API calls in the text sequences. The fine-tuning process ensures that the model not only retains its original language understanding but also learns when and how to incorporate external tools effectively. The API calls are integrated into the text in a way that aligns with the model's existing knowledge and patterns, preventing any negative impact on its overall performance.

We finetune M on C∗ using a batch size of 128 and a learning rate of 1 · 10^−5 with linear warmup for the first 10% of training.

6. What are the baselines used in the paper? What metric is used for the comparative study and how does Toolformer compare to them?
Ans: The baselines used in the Toolformer paper are:

GPT-J: A regular GPT-J model without any fine-tuning.

GPT-J + CC: GPT-J fine-tuned on a subset of CCNet without any API calls.

Toolformer (disabled): The same model as Toolformer, but with API calls disabled during decoding.

OPT (66B) and GPT-3 (175B): These are larger models used for comparison, approximately 10 and 25 times larger than the other baseline models, respectively.

The metric used for the comparative study is the performance improvement in various tasks, such as completing short statements with a missing fact. The results show that Toolformer clearly outperforms these baseline models, improving upon the best baseline by 11.7, 5.2, and 18.6 points respectively for different subsets. Notably, Toolformer also outperforms the much larger models, OPT (66B) and GPT-3 (175B), due to its ability to independently decide to ask the question-answering tool for the required information in almost all cases (98.1%). This significant improvement demonstrates the effectiveness of Toolformer in leveraging external tools to enhance its performance on tasks requiring factual knowledge.

7. What datasets are used in the paper? What tools are used by Toolformer for each dataset?
Ans: LAMA:SQuAD, Google-RE and T-REx subsets of the LAMA benchmark
Math Datasets:ASDiv(Miao et al., 2020), SVAMP (Patel et al., 2021) and the MAWPS benchmark (Koncel-Kedziorski et al.,2016).
Question answering: Web Questions (Berant et al., 2013), Natural Questions (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017).

We let the model choose its own tools whenever it feels and we do not give it information about which tools to use.
We disable tools like wikipedia search API for LAMA, question answering API for question answering, etc. to avoid giving it an unfair advantage.

8. What different ablation studies are done in the Toolformer paper?
Ans: In section 4.4 scaling laws, it investigates how Toolformer's performance changes with model size and its ability to utilize external tools. Experiments are conducted with GPT-J and smaller GPT-2 family models of varying sizes. The findings reveal that the capacity to effectively leverage external tools becomes significant at model sizes around 775M parameters. Smaller models show similar performance with or without tools, while larger models demonstrate improved ability to use these tools effectively.
Section 4.3 also checks if the toolformer degrades through fine tuning with API calls.

9. What is Langchain? What problem(s) does it address? How well (based on the internet searches) is it working?

Ans: Langchain is an open-source framework designed to develop applications powered by LLMs. It is primarily used for creating context-aware applications that can perform sophisticated reasoning, such as chatbots, virtual agents, and other LLM-driven applications. Langchain simplifies the integration of language models with external data sources, enabling applications to provide data-aware responses and make informed decisions based on user input

From a performance perspective, Langchain has shown significant improvements over traditional pipeline methods in tasks such as question answering, dialogue consistency, and common sense reasoning. Despite a moderate increase in latency, it maintains similar parameter counts while delivering better accuracy and consistency, making it a valuable tool for enhancing LLM applications. However, challenges remain, particularly in scaling to enterprise-grade applications due to issues like vector store retrieval latency and the need for specialized system design for handling large-scale queries

10. Do Langchain and Toolformer perform the same tasks? What are the key conceptual differences, if any?

Ans: LangChain helps in creating applications that require real-time information, context awareness, or interaction with external APIs and databases. It is used for developing question-answering systems, chatbots, interactive agents, and data summarization tools, among others.

The primary application of Toolformer is in enhancing the performance of language models across a variety of downstream tasks, such as zero-shot learning tasks, by integrating external tool capabilities directly into the language processing.

Key Conceptual Differences:

Integration vs. Autonomy: LangChain focuses on building applications that integrate LLMs with external data and tools, where the integration is primarily managed by the developer. In contrast, Toolformer autonomously decides when to use external APIs, with the integration built into the model itself.

Application Scope: LangChain is a framework for application development, providing tools and components for integrating LLMs into broader application contexts. Toolformer, meanwhile, is a specific model or approach within an LLM, enhancing its internal capabilities through external tools.

Developer Role: With LangChain, developers play a significant role in defining how the application interacts with external data and services. In the case of Toolformer, the model itself manages these interactions based on its training.

11. What questions do you have about Toolformer and Langchain?

Ans: The toolformer paper is very descriptive and clear in what they were trying to do. I am really interested in seeing Langchain in action as it is very interesting.