UNIVERSITÄT ZU LÜBECK

Using AutoGPT for Information Retrieval Agents

Nutzung von AutoGPT für Informations-Recherche Agenten

Masterarbeit

verfasst am
Institut für Informationssysteme

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Jakob Horbank

ausgegeben und betreut von
Prof. Dr. Ralf Möller

Lübeck, den 15. April 2024

IM FOCUS DAS LEBEN

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Jakob Horbank

Zusammenfassung

Es geht darum, der Welt »Hallo« zu sagen.

Abstract

It is about saying "hello" to the world.

Contents

1 Introduction

This is an introduction like not other. Information retrieval hard and stuff. Would be nice to have a chatbot that answer natural language questions and gives sources from research database and even web.

1.1 Contributions of this Thesis

Hopefully the above.

1.2 Related Work

There are different attempts at creating LLM outputs with sources. Perplexity AI hosts a question answering service, that gives source from websites.

Open Source LLM Agents

- AutoGPT
- babyagi

Information Retrieval Applications

In a variety of application contexts, the answers of an assistant have to be correct. This is especially true in research contexts. A model that hallucinates isn't feasable in this case. But while hallucination can be reducing with fine-tuning, it can not be competely eliminated. Perplexity AI is a online service that leverages a language model to provide a search that generates an answer from different sources in the internet. The content of the answer is then linked to the found sources, so the user can see and verify the result.

As this is a proprierty closed source product accessible through a web interface, this is not a useful to create our own research assistant. The provided API simply hosts different LLMs and allows prompting them. There is no possibilty of of fine-tuning.

1.3 Structure of this Thesis

I do this then that and then that.

2

Backgrounds

Different branches of research were used to build upon in this work. These topics and how their are connected to my work is explained in more detail in this section.

2.1 Information Retrieval

Retrieving the best documents for a query from a large databse, filled with information of different kind is a classic problem in computer science. There has been extensive reasearch on database architectures and indexing algorithms.

2.2 Agents

Although the notion of agent is not new, it recently gained attention in combination with the rise of generative language models.

2.3 Language Modeling

Natural language processing is a long studied research area of computer science. In recent years delevopments have significantly sped up, with the introduction of deep learning into NLP. The transformer archtiture has been the basis for all advancements in recent years.

Transformer Networks

Using recurrent structures such as RNNs was the dominant strategy in sequence modeling before transformer networks. Every sentence token was represented as a hidden state that is a function of all previous hidden states. While this approach has a reasonable motivation, the sequential nature contraints computation speed for a single training example. This limitation is especially hindering for longer sequences, where batching the training data is only possible to a memory limit. Attempts to minimize sequential computation included using convolutional networks that can be computed in parallel. For these models

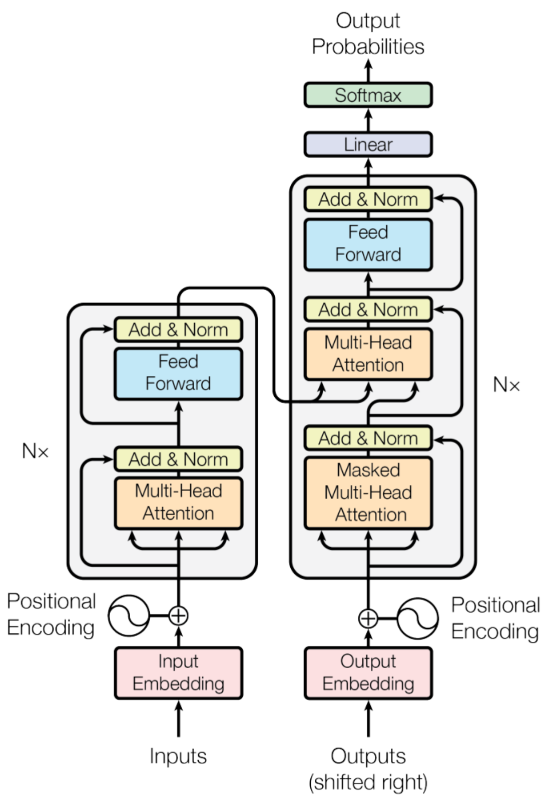

Figure 2.1: The encoder-decoder architecture of a full transformer network [4]. The encoder (left) can attend over all positions to learn rich embeddings. The decoder (right) can generate output sequences. The first decoder attention layer can only attend on previous position of the input sequence, while the second can attend over the outputs of the encoder. This combines a sequence-to-sequence with auto-regressive properties into the decoder.

however, the number of operations required to relate two tokens grows in the distance of their positions in the sequence. Attention mechanisms allow modeling token relationships independent of their distance in a sequence.

The transformer network [4] shown in Figure **??** was the first model that removes all recurrent structures and only relys on attention mechanisms. In particular, they use multi-headed self-attention layers. Attention mechanisms learn dependencies between tokens in sentences without regard to their distance. Their non-sequential charactericstics allow for better parallelization. Self-attention is an attention mechanism that computes relations between different positions of the same sequence with the goal of finding a representation of the sequence.

Since the transformer architecture does not use any recurrent structures, the order of the tokens in the sequence must be manually injected. Positional encodings are added to the input embeddings to achieve this.

Like most language modeling networks, a transformer consists of an enconder and a decoder. The encoder has two sub-layers, a multi-head attention block and a fully connected feed foward block. The decoder is similar to the encoder but has an additional

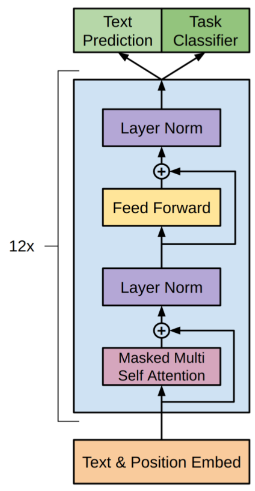

Figure 2.2: The GPT archictecture [3]. A transformer encoder without connections to a previous encoder is used. The attention layer can only attend on previous positions in the input sequence. The decoder is stacked 12 times before generating the final output.

multi-head attentention layer that attends on the outputs of the encoder. The first attention layer is masked and the decoder input sequence is shifted by right, so only previous positions can be attended by the decoder.

The full architecture has capabilites of a classic sequence-to-sequence model used for tasks like language translation. All positions in the decoder can attend over all positions in the encoder. For other tasks however, the individual encoder and decoder sections are a better fit, as explained in the sections **??** and **??**.

Generative Pre-Trained Transformers

Decoder only transformers make up the base of generative pre-trained transformers (GPT). The decoder used for GPT does not rely on the outputs of a encoder, because it designed made for generation tasks. In Figure **??** we can see that only the masked multi-head attention layer is kept in comparison to the the full transformer network. The simple archticture enables faster pre-training and fine-tuning.

GPT is trained in two phases. In the unsupervised pre-training phase, the model is trained with big datasets of unlabed text data. The training objective here is to accurataly generate the next token of a sequence.

After the first pre-training phase, the model needs to be fine-tuned a for a specific task. By including custom tokens in the fine-tuning dataset, the decoder can learn to handle a variety of different tasks such as text generation or classification.

Newer research on generation models focuses on scaling the network. The *scaling law* states that by simply increasing the amount of parameters of the network, the model capabilites increase. Furthermore, after a certain level of scaling, abilities emerge that the model was not directly trained on.

As large language model scaling has reached billion parameter counts, only very few companies have the hardware to train or even run inference tasks.

Embedding Models

The left-to-right nature of the transformer encoder suits text generation tasks that only depend on previous tokens. But there are dependencies between tokens in both directions, which means a decoder only model can not capture all relations.

The big amount of knowledge learned in the pre-training phase is encoded in embeddings. Embeddings are high dimensional vectors that represent a sequence of words or tokens. Documents that are semantically similar are represented with embedding vectors that have low distance, while semantically different documents have a high distance.

BERT [1] uses the encoder part of a transformer network, to create rich embeddings of input sequences. In pre-training, the bidrectional encoder is trainend with two unsupervised tasks. For the first task random token in the sequence are masked out and the objective is to predict the masked tokens from the remaining tokens in the sequence. In the second task, sentence level relationships are learned by predicting the next sentence. After pre-training the BERT model can be finetuned on different downstream tasks leveraging the richer bidirectional embeddings learned in pre-training.

Instruction-Tuned Models

Language models trained to predict the next token of a sequence. While alot of knowledge is captured in the weights of the model, most information in the internet is not formatted in conversatonal style. Because of this, langauge models need to be prompted in a specific way to be effective. The prompt has to be written in a way that the continuation of it yields the desired output. This is not optimal for human users, as they would rather write in a conversational style. Instruction tuned models solve that problem.

Instruct models are fine-tuned version of language models. Capturing the intent of the user is a key challenge for language models. This process is called *alignment*. popular approach to the alignment problem is reinforcement learning with human feedback (RLHF). Handcrafted prompts are used to fine-tune GPT-3. The outputs of the model are collected into a set and ranked by humans. This set is then used to train a reward model. With this reward model, the language model is further fine-tuned. The resulting model is called *InstructGPT* and performs better than the baseline GPT-3 model.

Large language models are trained to predict the next token of a sequence, not to follow the instruction of the user. This leads to some unwanted results, such as toxic, harmful answers or fabricated information that is not true.

3

Analysis of AutoGPT for Information Retrieval Tasks

AutoGPT is an open source project that tries to 'make GPT fully autonomous'. It started out as collection of loose scripts but quickly gained alot of interest in the open source community and grew into a much bigger and now funded project. Initally it only contained the AutoGPT agent that I will anaylse in this chapter but over time has seen additions such as a agent framework called *forge*, a benchmarking system that was put in place to host an agent hackathon as well as branching out of the project to create the *agent protocol*. The agent protocol is an attempt of the community to standardize agent applications.

The AutoGPT was not built with a focus on information retrieval, which is why I will analyse the information retrieval capabilites in this chapter.

GPT langauge models are used to control an agent that works towards reaching a stated goal. The project contains a core general purpose agent with a predefined set of abilities. Addidtionaly a baseline sdk is beeing developed to build custom agents.

3.1 Agent Backgrounds

The concept of agents in computer science is not new. An agent is a system that acts towards reaching a goal in an environment. Agents can be implemented as software or as physical robots or even humans. In the same way different envirmonments are possible such as the real physical world, a web browser or a simulation. The agent needs ways to sense its environment, which can be done by sensors in a physical environment or programatically in a software enivrnment. A task has to be specified for to agent so it knows what his goal is. It can then employ different strategies to reach that goal. These strategies consist of planning steps to execute. While acting out these steps, the environment will probably change as time passes, so an agent needs to reevaluate its plan and the contained steps.

The impressive capabilities of large language models have lead to research implementing them into agents system. In Zhou2023 a distinction between static and dynamic agents is proposed. A static agent is characterized as a fixed pipeline that mimics the users' behaviour. While this approach works, it cannot deal with complex and sometimes random human actions. The other type of agent can dynamically exectute actions that are presented it. The most prominent way of using LLMs for agents is to give build a prompt that contains all the information about the task, and the ask it to propose the next action. The prompt can contain descriptions of possible abilities, the task, guidelines, information about the environment and more.

3.2 Architecture Overview

The AutoGPT agent is modeled after the classic agent architecture. After the start, the user is asked to enter a task the AutoGPT agent should perform. Then the agent enters a loop of prompting the LLM, executing the proposed action and handling the result of the action and updating the agent state.

The LLM is prompted in a structured way. A base prompt template is defined and populated with current information before each prompting step. The information includes the task at hand, a list of possible actions, a history of previous actions and their results and some extra statements that are there to guide the language model. As the answer needs to be parsed, the system prompt defines a fixed format the LLM should answer in. The answer consists of the thoughts and the proposed next action.

AutoGPT is divided into four modules. The *brain* is the main module that controls the agent. In AutoGPT this is realized by prompting the language model in a structured way. Using the chat system prompt, the language model is prompted to answer a structred format. Different techniques are implemented in this structure. The language model is forced not only to plan the next step, but also to explain the choice for the chosen step and to add self-criticism. An extra output for the human user is also returned. The second part of the answer is the actual next action with the needed arguments. The action make up the second module of the agent. In this module the abilities of the agent are defined. These can be file operations, database queries or web search functionalities. The third module is the *memory*. Memories are modeled after humans which have short and long-term memory. Short term memory can be implemented as an in-memory list of messages to the language model. Long-term memory needs a persistent store such as a database. A popular option for language models are vector databases that work with embeddings.

Currently, the AutoGPT agent is implemented for OpenAI GPT models. The prompts are tailored in ways that benefit the charactericstics of GPT-3 and GPT-4. Switching to a different model is not diffult from a software perspective, but semantically poses a challenge. If one knows the message format for a specific model the input can be adjusted. But the same prompting techniques does not necessarily work for all langauge models. The challenge is to switch between different prompting styles, as every model needs to be prompted differently. For example, OpenAI models benefit from profile sentences like "You are an expert in computer science", while Anthropics Claude does not...

3.3 Information Retrieval Capabilities

The AutoGPT Agent has different abilities that can be utilized for information retrieval. Notably, it is able to search the web and operate on the file system.

The web search is implemented by a two-step process. First a search API like DuckDuckGo is called to get a list of relevant pages. Then the page contents are scraped with a headless browser. It is possible to read and write to text files. Other document types are processed by basic text extraction tools to get the plain text.

For longer files such as scientific journals the extracted text is too long for the language model. The AutoGPT agent has no ability to chunk the text into smaller chunk or store it in a database. This is a limitation that needs to be addressed for information retrieval tasks over a research database repository.

Having a vector database would enable techniques such as retrieval augmented generation. The agent would get a prompt which a question over the RDR and choose an action to start a semantic search over the vector database. The result of the search are the chunks that are semantically closest to the question. These chunks can then be included as context for the LLM prompt to generate an answer.

The default agent has a tendency towards searching the web for information. We want an agent that prioritizes information that is present in the research repository. This needs to be addressed in the prompting techniques of the agent.

4

Retrieval Augmented Generation Agent

Large language models are excellent at generating text in a variety of styles. But when prompted for about specific topics the answer quality suffers. Different approaches try to tackle this problem. A promising method is retrieval augmented generation, which combines in-context learning prompting techniques with a vector database. In this section I present an agent that is designed to answer user prompts in a retrieval augmented fashion.

4.1 Methods To Improve LLM Generation

Different approaches try to embed information sources into the generated text. One approach is to fine-tune the language model on a dataset that contains the information. The creation of fine-tuning data is an expensive task, as data needs to be gathered, checked a cleanup to produce good results in training. There is some emerging work on generating synthetic datasets by using large language models as data augmentation tools. But even if the fine-tuning data quality is sufficient, it is still a challenge to make a LLM an expert for a specific domain. Rather than learning new knowledge, fine-tuning is best suited to guide the model towards a certain answering style. Furthermore, fine-tuning billion parameter models is only possible on expensive hardware and therefore not feasible for on-demand tasks.

Following up on the challenges of fine-tuning researches have searched for ways to optimize prompts to get the best model performance. These methods make use of a phenomenon called in-context learning. In-context learning describes the observation that giving a large language model more context surrounding the prompt can drastically improve the response quality. [5] found that adding a sentence that suggests step-by-step thinking to solve a problem makes the model better at solving logical questions. For GPT models, [xzy] showed an improvement after giving the language model a hint that it is an expert in a certain topic. The findings in the area of prompting techniques and in-context learning suggests that prompt optimizations have a high potential of getting the most out of LLMs.

A more promising approach is retrieval augmented generation [2]. Before generating an answer to a prompt, a vector database is searched for relevant information. The prompt then includes the information of the returned documents as context.

- Fine-tuning
- Prompting techniques (Chain-of-thought, in-context learning)
- Retrieval augmented generation

4.2 Retrieval Augmented Generation

Large language models can embed a large amount of knowledge into the weights during the pre-training phase. It is possible to generate good answer for different kinds of prompts. ”write about what are good prompts for llms”. But when it comes to specific domain knowledge, the exactness of LLMs start to degrade.

When trying to prompt models about very specific topics common problems such as hallucationations start to show. Not only does the model generate wrong information, it does so with strong confidence. A promising method to make the model answer based on specific sources is called retrieval augmented generaton(RAG).

The RAG method combines an information store with in-context learning. A corpus of documents is stored in a database. When the user prompts the system the database is queried with that prompt matching documents are returned. These documents are then included in the prompt to the LLM as context.

To store the information a vector database is used. A vector database uses embedding models to embed each document into a high dimensional vectorspace. After a user query, the query string gets embedded by the same model, then a metric such as the cosine similarty is used to find semantically close documents. The top-k documents are then returned.

- what is a vectorestore compared to other databases
- how are documents stored and queried
- how is the information included in the prompt

4.3 Agent

To create the information retrieval agent, I used the Forge SDK[1] that is included in AutoGPT.

Forge SDK

To make collaboration on agents easier, the open source LLM-agent community created the agent protocol. The Agent Protocol defines an API schema that handles the communication with an agent. On a high level the protocol defines endpoints to create a task and to trigger the next step for the task.

The Forge SDK handles the boilerplate code that implements the agent protocol. On running, the server with the corresponding endpoints gets started and the agent can be

[1] https://docs.agpt.co/forge/get-started/

used over the API endpoints. AutoGPT also comes with a chatbot webapp that builds the appropriate HTTP requests to the agent endpoints.

The user of the Forge SDK has to create the actual agent logic, create custom prompt templates for the used model and add abilities to interact with external resources. Because the Forge SDK is still under development and by no means a polished product, some internals also have to be tweaked to achieve the desired agent behaviour.

By default the Forge agent comes with abilities to read and write textfiles and to search a search engine as well as scraping a webpage. The default behaviour is to write a test string into a file in the workspace no matter what the user prompts.

Agent Memory

Similar to the view of human memory, different implementations of memories for agents have been proposed. For short-term memory, message history format used by instruct language models can used. The last n messages of a conversation between the user and the agent are recored and change the generation behaviour of the LLM that controls the agent. Long-term memroy is generally represented as some external store where information can be written and stored.

– Vector Database
–

Agent Abilities

– Custom agent Loop
– Custom abilities
– Further guidances for the agent

5

Benchmarking for an IR Agent

The evaluation of large language model agents is a difficult task, as evaluating LLMs themselves presents a challenge. There are different approaches to evaluate systems built around language models. Subjective evaluation is based on human feedback. As LLM systems are generally made to serve humans this is an important part of evaluation. On the other hand, quantiative metrics that can be computed are used for objective evaluation. Different metrics are used for different tasks. Another importat method are benchmarks. Benchmarks are a set of tasks or an environment that the agent is to move in.

5.1 Existing IR Agent Benchmarks

As the space of possible agent domains is large, lots of different benchmarks were proposed. Simulation environments like

Although lots of benchmarks for LLM applications have been proposed, there are few benchmarks that are designed to test the information retrieval capabilities of an agent. There are some benchmarks that are used to evaluate information retrieval in general and in a diverse domains like ”cite”.

5.2 The AutoGPT Benchmarking System

To evaluate AutoGPT and other agent systems that implement the agent protocol, the AutoGPT project has implemented its own benchmarking system. The system consists of a set of tasks that the agent has to complete. The tasks are designed to test different aspects of the agent, and are divided into different topics. Some tasks depend on the previous succesful completion of other tasks. A task consists of an input prompt and a expected output. The ouput is defined by certain words that should be contained.

– Level based system
– Dependencies

5.3 Custom Benchmarks for Local IR over Journals

- Retrieval benchmarks

6

Results

6.1 Points of Failure

6.2 Benchmarking Results

6.3 Subjective Evaluation

- Why is subjective evaluation needed
- What aspects of the agent can be evaluated subjectively

7 Conclusion

- What was the premise
- What was tried out
- What worked what did not and why

7.1 Future Work

The reserach area about large language models currently moves at a rapid pace. While writing this thesis, communities have focus more on 'agentic' applications of LLMs. OpenAI has released OpenAI Assistants, which.... The AutoGPT team is still working on making the use of local models feasable.

Bibliography

[1] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of NAACL-HLT*. 2019.

[2] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.

[3] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. *Improving Language Understanding by Generative Pre-Training*. OpenAI, 2018.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention Is All You Need. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[5] Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 24824–24837.