Implementing Long-term Memory Management Using Graphs

Shubhankar Vakde

June 16/2023

1 Introduction

In this proposal, an "agent" is the voice-assistant that will use this memory management system to better interact with its users.

1.1 Disclaimer

Please note that this paper is simply an **experiment**, and might contain mistakes / wrong assumptions. After v1 is fully built out, I'll report on its results.

2 Data Structures

This system is implemented with Temporal Knowledge Graphs (TKGs). A TKG is similar to a regular graph, but with 3 important distinctions.

2.1 Arcs

TKGs use arcs, which are hierarchical connections between entities. Examples of categorized connections are ["likes", "dislikes", "contains", "distinct from"], and more. Using arcs allow TKGs to model a system of relationships to better understand its user, based on their daily interactions and conversations with the agent.

2.2 Relationship Strength

Using the arbitrary data (categories, timestamps, relations, etc.) that arcs can model between nodes, we can calculate weights between nodes that can be regarded as the strength of their relationships. When the agent queries data from the graph to respond to a user prompt, outputs with higher weight will be provided as context to the agent.

2.3 Time Decay Functions

Progressive decay of relationships are what replicate how people "forget" about things over time, and make sure the graph doesn't exceed size limits, and lead to inefficient queries. A function where a relationship starts at strength 1.0, and decays at a rate of 0.05 is modelled by the formula

$$f(t) = a \cdot \exp(-b \cdot t),$$

where a is the initial weight for the current decay period, b is the rate of decay and t is the time in seconds since last invocation, can update the TKG over time. An exponential function is used to model the rather unpredictable nature of human memory, but it is still insufficient to effectively emulate the difference between "working memory" and "long-term memory".

3 Implementation

To implement the TKG, there are a few options: NetworkX (Python library for building complex & dynamic networks), Neo4j (Graph DB designed for ML applications and supports sorted queries). Note that Neo4j is an full-on Database, and NetworkX is simply a library, and here are some pros and cons to each.

NetworkX is very flexible and allows for arbitrary data, but its gets worse with complex querying and long-term storage (we need to handle the infra for it). Neo4j has higher startup costs but over time, it will pay for itself due to its capacity for more complex and faster queries.

There are 2 parts to implementation: a) recall during conversation and b) updating the TKG after a conversation is over.

3.1 Recall during conversation

When given a prompt from a user, the agent uses a language like Cypher to query information from a users TKG to provide them better responses and keep context of the conversation. Constructing queries can be performed with another GPT code interpreter model fine-tuned for Cypher.

3.2 Updating the TKG

Updating a users TKGs after a conversation requires a longer and more involved process.

1. Given the transcript of a conversation, use a NLP library to execute NER (Named Entity Recognition) and generate a categorized list of important entities mentioned.
2. Use another NLP library to execute RE (Relation Extraction) on the transcript with the named entities.
3. Complete a traversal of the users TKG updating arcs where there is output from the RE on the conversation transcript, and update the weight for all the arcs using the time decay function.

Potential libraries to use are SpaCy, AllenNLP, and smaller models on Hugging Face.

4 Output Accuracy

There is a high probability of failure with this implementation because of automated and unpredictable nature of the various models used throughout the memory management process for the agent. It might misunderstand relations, incorrectly forget important things and outright make absurd connections.

I'll build it anyway, this seems pretty cool.