Personal Lessons From LLMs

@mattrickard

The brain metaphor for neural networks has always been a fun simplification but not a useful one under closer inspection. How we train and inference deep networks doesn't have much to do with how the human brain works.

But what if LLMs could teach us more about ways that we approach general reasoning and languages? Some personal lessons I've learned (or have been reinforced) from working with LLMs.

***What* you ask and *how* you ask a question really matter.** Prompt engineering is a sandbox showing us how different works and sequences change the result. Sometimes we're asking the wrong question. Other times, we're asking it ambiguously.

Style is hard to describe but a tangible thing. A Shakespeare play, a Rupi Kupor poem, a Monet painting. Some styles are apparent, but LLMs can surface unique styles in unexpected places.

Thomas Kinkade, **the painter of light**, dominated the Stable Diffusion styles in early models simply because he had thousands of paintings and a distinct (but maybe not artistically-revered) style.

Again, when we can remix prompts — e.g., contrasting a normal work email and the same written in the style of Shakespeare — we can start to extract some of the "rules of the style."

Thinking step by step leads to better results. A lesson from our adolescence that actually works. There's something about putting the process down on paper (or saying it aloud) that helps us solidify our reasoning.

Word analogies (embeddings) can help distill meaning. This is obvious for the usual examples (man :: woman, king :: queen, aunt :: uncle, kings :: queens). **Thinking geometrically** is a good approach.

Programming languages are still natural languages. LLMs significantly increased their reasoning ability (e.g., "chain-of-thought") **after being trained on code**. While programming languages are

much less ambiguous than the languages we speak, they still tell a story and leave room for opinionated choices (**why correctly naming your variables is so important**).

Sometimes the next choice is obvious; other times, it's not. LLMs let us see the next token probabilities for the current prompt sequence. For some situations, the next choice is unambiguous — a semicolon after a programming instruction or the token "time" after "once upon a ".

However, you can consider more uncommon choices by increasing the temperature parameter. It's interesting to think through problems with this lens — when do you want to make a creative choice? When do you want the logical or obvious choice?

Breaking a big problem into subproblems is often a great strategy. Asking an LLM to solve a complex problem or write an intricate program is bound to fail. Even with chain-of-thought prompting, these models can easily get sidetracked. Instead, they are most effective when asked to solve smaller problems. Things that can fit inside a reasonable context window — e.g., a small function (rather than an entire program) or a paragraph of a book (rather than a whole book).