In-context learning: what is a prompt expected to do ?

Consider k exemplars

$$\langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \ldots, \langle \mathbf{x}^{(k)}, \mathbf{y}^{(k)} \rangle$$

These exemplars need to be encoded into a single *context* $\dot{\mathbf{x}}$ amenable to a model solving text-continuation ("predict the next").

For example

$$\begin{aligned} \dot{\mathbf{x}} = \mathrm{concat}(\quad & \mathbf{x}^{(1)}, \langle \mathrm{SEP}_1 \rangle, \mathbf{y}^{(1)}, \langle \mathrm{SEP}_2 \rangle, \\ & \vdots \\ & \mathbf{x}^{(k)}, \langle \mathrm{SEP}_1 \rangle, \mathbf{y}^{(k)}, \langle \mathrm{SEP}_2 \rangle, \\ & \mathbf{x} \\ &) \end{aligned}$$

Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm (https://arxiv.org/pdf/2102.07350.pdf)

But what is the role of the exemplars ?

Our initial supposition

- to *demonstrate* a new Target Task by giving the feature/label mapping relationship
- *meta-learning*

Yet the paper [Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm (https://arxiv.org/pdf/2102.07350.pdf)](https://arxiv.org/pdf/2102.07350.pdf) demonstrates that

- increasing k (adding more exemplars) sometimes *hurts* performance
- keeping k fixed
    - the exact form of the context affects performance

This is inconsistent with the meta-learning supposition.

They propose a new theory about the context's role

- to locate a task *learned in pre-training*

They offer suggestions on crafting prompts according to this theory

Note

This is a theory, not proven fact.

Nonetheless, the suggestions for prompt engineering

- are interesting
- may lead to better performance.

We present the suggestions in turn

Signifier: direct specification

The theory is that the LLM has learned certain behaviors (functions ?) *during pre-training*.

In order to get the LLM to perform the task at inference time

- we need to use a *signifier*
- block of text that has become associated with the desired behavior
- a **zero shot** prompt

Unlike an examplar

- it specifies *what* the Target Task is
- not *how* to perform it

So Prompt Engineering in this theory is discovering the signifier (name of the function ?) for the desired task.

For example, for a Target Task that translates from French to English.

The following contexts uses a direct form (guessed) of the signifier

```
French sentence is <French phrase>. Translate from French to English.
```

- n.b., we use descriptions bracketed by < and > as place-holders for user-supplied values.

Signifier: via demonstration

The hypothesis is that the LLM has essentially learned "parameterized functions" through examples.

Exemplars are

- instances of the function being called
- with different formal parameters

Here we provide a demonstration of the function

- similar to the meta-learning theory
- but with the objective of invoking a task learned in pre-training

```
French: <French phrase 1>
English: <English translation 1>

...
French: <French phrase k>
English: <English translation k>

French: <source phrase>
English:
```

Constraining behavior

Remember, the LLM was trained in the text-continuation task (predict the next).

The result of the continuation may be inconsistent with our intent but be a valid continuation anyway.

Consider the following context

```
Translate the following French sentence to English.
<source phrase>
```

The LLM might continue with more French

- continuing the thought of `<source phrase>` in French

Adding *syntactic constraints* to the context may invoke behavior more consistent with the Target Task.

- Adding delimiters
- That might be the purpose of `French`, `English` and the newline character in

```
French: <French phrase 1>
English: <English translation 1>
```

- rather than being an actual *demonstration*

Imitation

This seems to be equivalent to Role Playing

- achieving the desired behavior by requesting the LLM "imitate" a subject-area expert

Rather than specifying *how* to perform a Target Task

- invoke an expert to imitate

```
A French phrase is provided: <source_phrase>
The masterful French translator flawlessly
translates the phrase into English:
```

In [ ]:
```
print("Done")
```