Reverse-engineering LLMs: making the most of the context
Summary

Mathis Embit

August 20, 2024

Abstract

Here is a summary of my work on prompt optimization. The work start by observing LLMs sensitivity to the input and leads to constructing adversarial prompt attacks. To do so we study discrete and continuous prompt optimization and find a way to translate continuous embeddings to discrete tokens in a meaningful way.

1 LLMs are sensitive to the context

Observation: few-shot learning [1]. We want to understand the influence of the input on the output. We look at gradient of the output with respect to the input to assign each output token an input tokens explaining it (as in grad-cam [6]). Leveraging the context of LLMs to improve their performance is called in-context learning. A subfield of in-context leanring is prompt optimization. For a given task dataset we use the output gradient wrt the input to make the input perform better on the task. This optimization can be done at the token level (discrete optimization) or at the embedding level (continuous optimization). Contrary to continuous optimization, the discrete one is interpretable.

2 Discrete optimization

A first idea we are not interested in is to use the reasoning capacity of LLMs to let it improve the prompt iteratively by itself. This is too experimental to answer our question. Instead we use the gradient of the output wrt the input to swap tokens and reach our target. Shin et al. [7] use the gradient of the loss wrt the input to swap a given token by the one which is the less colinear with the gradient. Later Zou et al. [8] proposed an improvement by adding some randomness in the swaps, which improve the performance. Here are some notations and the two algorithms:

- x_{inp}: input of the user, which we should not modify
- $\mathcal{I}$: indices of the triggers tokens
- $x_{\mathcal{I}}$: triggers tokens we optimize
- x_{ins}, x'_{ins}: instructions of the aligned LLM (a meta-prompt)
- $x_{1:n} = x_{ins}x_{inp}x_{\mathcal{I}}x'_{ins}$: final prompt sent to the LLM
- w: embedding of a token from the vocabulary $\mathcal{V}$
- y: target (can be multiple tokens)
- $L(x_{1:n}) = -\log p(y|x_{1:n})$: the loss function

Algorithm 1 AutoPrompt

for $t = 1$ **to** nb_steps **do**
  for $i \in \mathcal{I}$ **do**
    for $w \in \mathcal{V}$ **do**
      compute $w^T.\nabla_{x_i} L(x_i)$
    end for
    $\mathcal{X}_i = \underset{w \in \mathcal{V}}{\text{top-k}} \left(w^T.\nabla_{x_i} L(x_i)\right)$
    $x_i = \underset{w_{\text{cand}} \in \mathcal{X}_i}{\arg\min}\, L(w_{\text{cand}})$
  end for
end for

Algorithm 2 Greedy Coordinate Gradient

for $t = 1$ **to** nb_steps **do**
  for $i \in \mathcal{I}$ **do**
    for $w \in \mathcal{V}$ **do**
      compute $w^T.\nabla_{x_i} L(x_i)$
    end for
    $\mathcal{X}_i = \underset{w \in \mathcal{V}}{\text{top-k}} \left(w^T.\nabla_{x_i} L(x_i)\right)$
  end for
  for $b = 1, \ldots, B$ **do**
    $\tilde{x}_{1:n}^{(b)} := x_{1:n}$
    $i \sim \mathcal{U}(\mathcal{I})$
    $\tilde{x}_i^{(b)} \sim \mathcal{U}(\mathcal{X}_i)$
  end for
  $b^* = \underset{b}{\arg\min}\, L(\tilde{x}_{1:n}^{(b)})$
  $x_{1:n} = \tilde{x}_{1:n}^{(b^*)}$
end for

3 Continuous optimization

Using additionnal embeddings to the input belongs to Parameter-Efficient Fine-Tuning (PEFT) methods. There are several way to do so (Lester et al. [3], Li and Liang [4], Liu et al. [5]). We will focus on the following basic gradient descent algorithm:

Algorithm 3 Continuous optimization of the trigger embeddings

for $t = 1$ **to** nb_steps **do**
  for $i \in \mathcal{I}$ **do**
    $x_i = x_i - \text{lr}.\nabla_{x_i} L(x_i)$
  end for
end for

Practically continuous optimization works better. Continuous embeddings offer more expressivness and we tend to easily fall into local minima. However continuous prompts are not interpretable. The first thing we can think of is to project (L^2, dot product, cosine, ...) the continuous embeddings on the embedding matrix so we can assign to each embedding a token in the vocabulary. Unfortunately this does not work (Khashabi et al. [2]), probably because the loss function is not smooth at all and the gradient descent can find a lot of local minima (maybe because LLMs are overparametrized and they are trained on real tokens, not continuous embeddings). Hence we need to find another way to go from continuous embeddings to discrete tokens.

4 From continuous embeddings to discrete tokens

As continuous optimization is powerful we want to leverage it before selecting discrete tokens. Rather than trusting the embedding space, we will opt for a probabilistic point of view. Given a continuous embedding we find its projection on the vocabulary by searching for the vocabulary token that has the most similar effect on the output of the LLM. More precisely we want to find the vocabulary token that minimize a divergence between its output and the output given the continuous embedding as input. Let's call this projection the forward projection (because we need to make a forward pass through the LLM). Formally:

L^2 projection	$\text{Proj}(x_i) = \underset{w \in \mathcal{V}}{\arg\min} \lVert x_i - w \rVert_2$
Dot product projection	$\text{Proj}(x_i) = \underset{w \in \mathcal{V}}{\arg\max} \langle x_i, w \rangle$
Cosine projection	$\text{Proj}(x_i) = \underset{w \in \mathcal{V}}{\arg\max} \frac{\langle x_i, w \rangle}{\lVert x_i \rVert . \lVert w \rVert}$
Forward projection	$\text{Proj}(x_i) = \underset{w \in \mathcal{V}}{\arg\min} \tilde{\mathcal{D}}(p(x) \lVert p(x)(x_i \leftarrow w))$

where, given an embedding $x = x_{1:n}$,

- $p(x)$ is the collection of the m LLM output distributions
- $\mathcal{D}$ is a divergence

- we write $\tilde{\mathcal{D}}(p(x)\|p(x)(x_i \leftarrow w)) = \sum_{j=1}^{m} \mathcal{D}(p(x)_j\|p(x)(x_i \leftarrow w)_j)$ the sum of the m divergences

Then a new algorithm could be:

1. Continuous optimization gives us $x_1^*, \dots, x_n^*$ the continuous solution to $\underset{x_{1:n} \in (\mathbb{R}^d)^n}{\arg\max}\ p(y_{1:m}|x_{1:n})$.
2. Discrete optimization of $x_1, \dots, x_n$ with the $\tilde{\mathcal{D}}$ loss to find $\underset{w_1,\cdots,w_n \in \mathcal{V}^n}{\arg\min} \sum_{i=1}^{n} \tilde{\mathcal{D}}(p(x)\|p(x)(x_i \leftarrow w_i))$

However the first step might converge to a solution such that $p(y_{1:m}|x^*_{1:n}) \approx 1$. Let's look at the distribution of the first generated token. If we write $p = p(x)$ its distribution with the embeddings as input and $q^{(x_i \leftarrow w)} = p(x)(x_i \leftarrow w)$,

$$\begin{aligned}
\min_{w \in \mathcal{V}} \mathcal{D}(p\|q^{(x_i \leftarrow w)}) &\iff \min_{w \in \mathcal{V}} \sum_{v \in V} p(v) \log \frac{p(v)}{q^{(x_i \leftarrow w)}(v)} \\
&\iff \min_{w \in \mathcal{V}} -\sum_{v \in V} p(v) \log q^{(x_i \leftarrow w)}(v)
\end{aligned}$$

If $p(y) = 1$, $\iff \min_{w \in \mathcal{V}} -\log q^{(x_i \leftarrow w)}(y) = \min_{w \in \mathcal{V}} -\log p(y|x_1, \dots, w, \dots, x_n)$. And in this case it does not change anything from the previous discrete algorithms. In order to make the divergence useful we need $p(y_{1:m}|x^*_{1:n})$ not to be peaked on the target y. To do so we can think of another loss for the first continuous optimization:

$$\mathcal{L}(x_i) = -\log p(y|x) + H(\text{softmax}(Ex_i)) - \log p(x) - H(p(.|x))$$

with

- $-\log p(y|x)$ the original cross entropy loss so it generates y
- $H(\text{softmax}(Ex_i))$ to attract x_i toward one token of the embedding matrix E (and repulse it from the others)
- $-\log p(x)$ the negative log likelihood of the virtual embeddings so that when minimizing it the corresponding discrete tokens will be more interpretable
- $-H(p(.|x)) = -\sum_{i=1}^{m} H(p(.|x, y_{<i}))$ an entropy regularization term so that the divergence trick will be useful

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

[2] Daniel Khashabi, Shane Lyu, Sewon Min, Lianhui Qin, Kyle Richardson, Sean Welleck, Hannaneh Hajishirzi, Tushar Khot, Ashish Sabharwal, Sameer Singh, and Yejin Choi. Prompt waywardness: The curious case of discretized interpretation of continuous prompts, 2022. URL https://arxiv.org/abs/2112.08348.

[3] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021. URL https://arxiv.org/abs/2104.08691.

[4] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021. URL https://arxiv.org/abs/2101.00190.

[5] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too, 2023. URL https://arxiv.org/abs/2103.10385.

[6] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, October 2019. ISSN 1573-1405. doi: 10.1007/s11263-019-01228-7. URL http://dx.doi.org/10.1007/s11263-019-01228-7.

[7] Taylor Shin, Yasaman Razeghi, Robert L. Logan IV au2, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts, 2020.

[8] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023.