Prompt optimization for a given task

Mathis Embit

May 17, 2024

Abstract

Here is a short explanation on how to optimize a single prompt to achieve a given task (e.g. sentiment analysis, natural language inference). We draw inspiration from AutoPrompt (Shin et al. [1]) and Universal attacks (Zou et al. [2]) papers.

1 Optimizing for a task

Now we would like to have single optimized suffix that would work for every instance of a given task. To optimize for a specific task we have several possibilities:

- Optimize the same trigger tokens example after example. The risk is that when we finish optimizing it does not work anymore with the first examples. Hence it is probably not a good choice.
- Optimize over every examples at the same time by minimizing the sum of every loss.
- Incorporate examples incrementally. When the optimization succeeds for a current example we add another one by adding its loss to the sum of the losses.
- Maybe there exist better methods.

AutoPrompt paper [1] does not detail how they optimize over the dataset of a given task. They only say that

> "The trigger tokens are iteratively updated to maximize the label likelihood **over batches of examples**."

Universal attacks paper [2] explains how to handle a dataset of multiple inputs with a single optimized suffix like this:

> "We find that incorporating new prompts incrementally, only after identifying a candidate that works as an adversarial example for earlier ones, yields better results than attempting to optimize all prompts at once from the start."

Here is the universal prompt optimization algorithm from Zou et al. [2]:

Algorithm 1 Universal Prompt Optimization

Require: Prompts $x^{(1)}_{1:n_1} \dots x^{(m)}_{1:n_m}$, initial suffix $p_{1:l}$, losses $\mathcal{L}_1 \dots \mathcal{L}_m$, iterations T, k, batch size B
  $m_c := 1$ ▷ Start by optimizing just the first prompt
  loop T times
    for $i \in [0 \dots l]$ **do**
      $\mathcal{X}_i := \text{Top-}k(-\sum_{1 \le j \le m_c} \nabla_{e_{p_i}} \mathcal{L}_j(x^{(j)}_{1:n} \| p_{1:l}))$ ▷ Compute aggregate top-k substitutions
    end for
    for $b = 1, \dots, B$ **do**
      $\tilde{p}^{(b)}_{1:l} := p_{1:l}$ ▷ Initialize element of batch
      $\tilde{p}^{(b)}_i := \text{Uniform}(\mathcal{X}_i)$, where $i = \text{Uniform}(\mathcal{I})$ ▷ Select random replacement token
    end for
    $p_{1:l} := \tilde{p}^{(b^\star)}_{1:l}$, where $b^\star = \arg\min_b \sum_{1 \le j \le m_c} \mathcal{L}_j(x^{(j)}_{1:n} \| \tilde{p}^{(b)}_{1:l})$ ▷ Compute best replacement
    if $p_{1:l}$ succeeds on $x^{(1)}_{1:n_1} \dots x^{(m_c)}_{1:n_m}$ and $m_c < m$ **then**
      $m_c := m_c + 1$ ▷ Add the next prompt
    end if
  end loop
Ensure: Optimized prompt suffix p

m_c is a counter that indicates how many examples we should use to compute the loss. The loss we minimize corresponds to the sum of the losses on the m_c first examples. We note that for each example $x^{(j)}_{1:n}$ we concatenate the same trigger tokens $p_{1:l}$.

Then, as in the optimization for a single generation, we randomize the choice of the candidate and its position in the trigger tokens.

Finally if $p_{1:l}$ succeeds in generating the solution for the first m_c examples, we add another example by incrementing m_c.

2 New algorithm

We will keep the AutoPrompt $\nabla_{x_i} L(x_i)$ gradient notation and use the Universal attacks algorithm. Randomizing the swap works better in practice and adding examples incrementally seems to be the best choice for using the task dataset.

Algorithm 2 Task Prompt Optimization

Require: Training dataset $\left\{\left(x^{(1)}_{1:n_1}, y_1\right), \dots, \left(x^{(m)}_{1:n_m}, y_m\right)\right\}$, initial trigger tokens $p_{1:l}$, losses $L_1 \dots L_m$, number of iterations T, k, batch size B
  $m_c := 1$
  loop T times
    for $i \in \{1 \dots l\}$ **do**
      $\mathcal{X}_i := \underset{w \in \mathcal{V}}{\text{top-k}} \left(\sum_{j=1}^{m_c} w^T . \nabla_{p_i} L_j(p_i)\right)$ where $L_j(p_i) = \log p(y_j | x^{(j)}_{1:n_j} \cup p_{1:l})$
    end for
    for $b = 1, \dots, B$ **do**
      $\tilde{p}^{(b)}_{1:l} := p_{1:l}$
      $\tilde{p}^{(b)}_i := \text{Uniform}(\mathcal{X}_i)$, where $i = \text{Uniform}(\mathcal{I})$
    end for
    $p_{1:l} := \tilde{p}^{(b^\star)}_{1:l}$, where $b^\star = \arg\min_b \sum_{j=1}^{m_c} L_j(x^{(j)}_{1:n_j} \cup \tilde{p}^{(b)}_{1:l})$
    if $p_{1:l}$ succeeds on $x^{(1)}_{1:n_1}, \dots, x^{(m_c)}_{1:n_{m_c}}$ and $m_c < m$ **then**
      $m_c := m_c + 1$
    end if
  end loop
Ensure: Optimized trigger tokens p

Sometimes the label y is more complex than a single word of the vocabulary. For example if the label y consists in several words $\mathcal{V}_y$ we need to marginalize the likelihood like this:

$$p(y|x_{1:n} \cup p_{1:l}) = \sum_{w \in \mathcal{V}_y} p(w|x_{1:n} \cup p_{1:l})$$

3 Tasks and evaluation

Some datasets:

- https://trojandetection.ai/
- https://www.kaggle.com/competitions/kaggle-llm-science-exam
- https://sites.ualberta.ca/~rabelo/COLIEE2021/ (used in https://arxiv.org/pdf/2212.01326)

References

[1] 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.

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