AutoPrompt vs Greedy Coordinate Gradient

Mathis Embit

May 16, 2024

Abstract

We detail two prompt optimization algorithms: AutoPrompt (Shin et al. [2]) and Greedy Coordinate Gradient algorithms (Zou et al. [3]). We show off the differences between the two by unifying the two papers notations, which is especially useful for the gradient computation explanations.

1 Algorithms

1.1 AutoPrompt

Notations:

- x_{inp}: input of the user, which we should not modify
- $x_{trig} = [T] \dots [T]$: triggers tokens we optimize
- $x_{prompt} = \lambda(x_{inp}, x_{trig}) = x_{inp}[T] \dots [T][MASK]$: final prompt sent to the LLM
- w: embedding of the corresponding token $w \in \mathcal{V}$

Algorithm 1 AutoPrompt

for $t = 1$ **to** nb_steps **do**
    for $x_{trig}^{(j)} \in x_{trig}$ **do**
        for $w \in \mathcal{V}$ **do**
            compute $\nabla_{x_{\text{trig}}^{(j)}} \log p \left(y \middle| \left\{ x_{\text{prompt}},\ \text{where } x_{\text{trig}}^{(j)} = w \right\} \right)$
        end for
        $\mathcal{V}_{\text{cand}} = \underset{w \in \mathcal{V}}{\text{top-k}} \left(w^T . \nabla_{x_{\text{trig}}^{(j)}} \log p(y | x_{\text{prompt}}) \right)$
        $x_{\text{trig}}^{(j)} = \underset{w_{\text{cand}} \in \mathcal{V}_{\text{cand}}}{\text{argmax}}\ p \left(y \middle| \left\{ x_{\text{prompt}},\ \text{where } x_{\text{trig}}^{(j)} = w_{\text{cand}} \right\} \right)$
    end for
end for

1.2 Greedy Coordinate Gradient

Notations:

- x_{ins}, x'_{ins}: instructions of the aligned LLM (a meta-prompt)
- x_{inp}: input of the user, which we should not modify
- x_{trig}: triggers tokens we optimize
- $x_{prompt} = x_{ins} x_{inp} x_{trig} x'_{ins}$: final prompt sent to the LLM
- $\mathcal{I}$: indices of the triggers tokens ($x_{trig} = x_{\mathcal{I}}$)
- $\mathcal{L} : x_{1:n} \in \{1, \dots, V\}^n \mapsto -\log p(x^*_{n+1:n+H} | x_{1:n}) \in \mathbb{R}$: the loss function
- $e_{x_i} = [0 \dots 010 \dots 0] \in \{0, 1\}^V$: one-hot encoding of the token $x_i \in \{1, \dots, V\}$
- nb_steps: number of gradient steps

Algorithm 2 Greedy Coordinate Gradient

for $t = 1$ **to** nb_steps **do**
  for $i \in \mathcal{I}$ **do**
    compute $-\nabla_{e_{x_i}} \mathcal{L}(x_{1:n})$
    $\mathcal{X}_i = \underset{w \in \{1,\ldots,V\}}{\text{top-k}} - \nabla_{e_{x_i}} \mathcal{L}(x_{1:n})$
  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
  $b^* = \underset{b}{\text{argmin}} \mathcal{L}(\tilde{x}_{1:n}^{(b)})$
  $x_{1:n} = \tilde{x}_{1:n}^{(b^*)}$
end for

2 Common structure

In order to compare the two algorithms let's replace:

- x_{prompt} by $x_{1:n}$ (and remove $[MASK]$ because it's quite confusing)
- $x_{trig}^{(j)} \in x_{trig}$ by $i \in \mathcal{I}$
- $\log p(y|x_{1:n})$ by $L(x_i)$ when dealing with the i-th token
- $\nabla_{e_{x_i}} L(w)$ by $w^T.\nabla_{x_i} \log L(x_i)$
- $x_{n+1:n+H}^*$ by y
- $\{1, \ldots, V\}$ by $\mathcal{V}$

Algorithm 3 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
    $\mathcal{X}_i = \underset{w \in \mathcal{V}}{\text{top-k}} \left(w^T.\nabla_{x_i} L(x_i)\right)$
    $x_i = \underset{w_{\text{cand}} \in \mathcal{X}_i}{\text{argmax}} L(w_{\text{cand}})$
  end for
end for

Algorithm 4 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
  $b^* = \underset{b}{\text{argmax}} \log p(y|\tilde{x}_{1:n}^{(b)})$
  $x_{1:n} = \tilde{x}_{1:n}^{(b^*)}$
end for

3 Gradient computation

Now that we cleared up the notations let's unveil the gradient computation. But first, why do we need to compute a gradient?

If we want an exact solution when dealing with a finite number of possible values we can simply compute the function output for every possible change in the variable to be optimize. In our case we want to change the i-th token x_i of the prompt and we have a vocabulary of tokens $\mathcal{V} = \left\{w_1, ..., w_{|\mathcal{V}|}\right\}$. We want to find the best token to swap with the current x_i. We could have just computed

$$\begin{bmatrix} p\left(y|x_{1:n}\right) \text{ where } x_i = w_1 \\ p\left(y|x_{1:n}\right) \text{ where } x_i = w_2 \\ \vdots \\ p\left(y|x_{1:n}\right) \text{ where } x_i = w_{|\mathcal{V}|} \end{bmatrix}$$

and picked the token resulting in the highest probability. However this requires $|\mathcal{V}|$ forward pass. Considering the fact that w is a token, i.e. a subset of a word, $|\mathcal{V}|$ is very large (even if we only consider character swap and the alphabet, 26 forward pass is costly).

3.1 AutoPrompt

How to compute $\nabla_{x_i} \log p\left(y|x_{1:n}\right)$?

Let's write $L(x_1, \ldots, x_n) = \log p\left(y|x_{1:n}\right)$ the loss we want to maximize, given a token y to generate.

If we write $\tilde{x}_i$ the token x_i after the swap we want to maximize $L(\tilde{x}_i)$ $(= L(x_1, \ldots, \tilde{x}_i, \ldots, x_n))$ over the possible $\tilde{x}_i$.

Aproximation: $L(\tilde{x}_i) \approx L(x_i) + \langle \nabla_{x_i} L(x_i), \tilde{x}_i - x_i \rangle = \langle \nabla_{x_i} L(x_i), \tilde{x}_i \rangle + L(x_i) - \langle \nabla_{x_i} L(x_i), x_i \rangle$

So $\underset{\tilde{x}_i \in E}{\operatorname{argmax}} L(\tilde{x}_i) \approx \underset{\tilde{x}_i \in E}{\operatorname{argmax}} \langle \nabla_{x_i} L(x_i), \tilde{x}_i \rangle$

We derive by the token embeddings hence it's a classic derivative with respect to continuous variable. We can apply the chain rule and the computation requires one forward and one backward pass of the model.

Let's write D the embedding size. $\nabla_{x_i} L(x_{1:n}) \in \mathbb{R}^D$ gives us the exact direction in the embedding space that increases the most the loss when swapping the i-th token.

We cannot directly make a gradient step because it would gives us an embedding that will probably not be the one of an existing token. Instead we should evaluate the gradient in each $w \in \mathcal{V}$: $w^T.\nabla_{x_i} \log p(y|x_{1:n})$. This costs $|\mathcal{V}|$ times the complexity of one projection layer. It informs us on the likelihood increase when replacing the i-th token by w. As we said we can think of this as the swapping making $L(\tilde{x}_i)$ (or the gap $L(\tilde{x}_i) - L(x_i)$) increase the most. We can also think of choosing the token of the vocabulary that is the most colinear with the gradient.

Comment: what about making a gradient ascent and at each step, before evaluating the new current value of the function take as the iterate the closest token in the embedding space? Is it the same? a simple projection? What if we don't pick the closest one at each step and wait for convergence?

3.2 Greedy Coordinate Gradient

How to compute $\nabla_{e_{x_i}} \log p\left(y|x_{1:n}\right)$?

In fact we will not derive by the token one-hot encoding because it's a discrete variable. The computation will be exactly the same as in AutoPrompt, we just need to exhibit the embedding matrix multiplication.

If we write $\Phi \in \mathbb{R}^{D \times V}$ the embedding matrix (one column = one embedding), we get the embedding x of a one-hot encoded e by performing the multiplication $x = \Phi.e$.

We have:

$$\begin{array}{ccccccc} \{0,1\}^{|\mathcal{V}|^n} & \longrightarrow & \mathbb{R}^{D^n} & \xrightarrow{\text{LLM}} & \mathbb{R}^D & \longrightarrow & \mathbb{R} \\ (e_1, \ldots, e_n) & \longmapsto & (x_1, \ldots, x_n) & \longmapsto & x_{n+1} & \longmapsto & L(e_{1:n}) = L(x_{1:n}) \end{array}$$

At index $i \in \mathcal{I}$, when we are optimizing over x_i/e_i, L only depends on x_i/e_i so we write:

$$\begin{array}{ccccccc} \{0,1\}^{|\mathcal{V}|} & \longrightarrow & \mathbb{R}^{D} & \xrightarrow{\text{LLM}} & \mathbb{R}^D & \longrightarrow & \mathbb{R} \\ e_i & \longmapsto & x_i & \longmapsto & x_{n+1} & \longmapsto & L(e_i) = L(x_i) \end{array}$$

Let's write

$$h : e_i \in \{0,1\}^{|\mathcal{V}|} \longmapsto \Phi e_i = x_i \in \mathbb{R}^D$$

and

$$g : x_i \in \mathbb{R}^D \longmapsto L(x_i) \in \mathbb{R}$$

$L = g \circ h$ so $dL(e) = dg(x).J_h(e)$ and $\nabla_e L(e) = J_h(e)^T.\nabla_x g(x) = \Phi^T.\nabla_x g(x)$.

Finally, by <u>approximating</u> $\tilde{e_i} \in \{0,1\}^{|\mathcal{V}|}$ with $\tilde{e_i} \in \mathbb{R}^{|\mathcal{V}|}$, we have:

$$\begin{aligned}
\underset{\tilde{e_i} \in \mathbb{R}^{|\mathcal{V}|}}{\operatorname{argmax}} L(\tilde{e_i}) &\approx \underset{\tilde{e_i} \in \mathbb{R}^{|\mathcal{V}|}}{\operatorname{argmax}} \langle \nabla_{e_i} L(e_i), \tilde{e_i} \rangle \\
&= \underset{\tilde{e_i} \in \mathbb{R}^{|\mathcal{V}|}}{\operatorname{argmax}} \langle \Phi^T \nabla_{x_i} L(x_i), \tilde{e_i} \rangle \\
&= \underset{\tilde{e_i} \in \mathbb{R}^{|\mathcal{V}|}}{\operatorname{argmax}} \tilde{e_i}^T \Phi^T \nabla_{x_i} L(x_i) \\
&= \underset{\tilde{e_i} \in \mathbb{R}^{|\mathcal{V}|}}{\operatorname{argmax}} \langle \nabla_{x_i} L(x_i), \Phi \tilde{e_i} \rangle \\
&= \underset{\tilde{x_i} \in \mathbb{R}^{D}}{\operatorname{argmax}} \langle \nabla_{x_i} L(x_i), \tilde{x_i} \rangle
\end{aligned}$$

$\nabla_{e_i} L(e_i) \in \mathbb{R}^{|\mathcal{V}|}$ directly gives us how much the likelihood increases when replacing the i-th token by every possible token of the vocabulary $\mathcal{V}$. However the gradient computation is strictly the same as AutoPrompt.

Comment: exactly the same thing is done in HotFlip (Ebrahimi et al. [1]).

References

[1] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification, 2018.

[2] 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.

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