Extension of Weight-of-Evidence Attribution Score to Language Modeling

Assume access to $\log p(x) = g_\theta(x)$, where g_θ is GPT-2.

$$\begin{aligned}\log p(x) &= \sum_{t=1}^{T} \log p_\theta(x_t \mid x_{1:t-1}) \\ \operatorname{logodds}(p(x)) &= \log \frac{p(x)}{\log(1-p(x))} \\ &= \log p(x) - \log(1 - \exp \log p(x))\end{aligned}$$

We also have access to the logits of each call of the model, i.e.

$$\begin{aligned}z_t &= \operatorname{logodds}(p_\theta(x_t \mid x_{1:t-1})) \\ &= \log p_\theta(x_t \mid x_{1:t-1}) - \log(1 - \exp \log p_\theta(x_t \mid x_{1:t-1})),\end{aligned}$$

but it's not super clear to me that that is helpful. Say we marginalize this sentence over the token at index i, and call the marginal probability $p_i(x)$. Assume $p(w|x_{1:i-1}, x_{i+1:T})$ is given be the masked language model (MLM) of BERT, whereas $p(x_{1:i-1}, w, x_{i+1:T})$ can be computed by GPT-2 (by replacing the i-th token of x with word w).

$$\begin{aligned}p_i(x) &= \sum_{w \in \mathcal{V}} p(x_{1:i-1}, w, x_{i+1:T}) \cdot p(w \mid x_{1:i-1}, x_{i+1:T}) \\ \log p_i(x) &= \log \sum_{w \in \mathcal{V}} p(x_{1:i-1}, w, x_{i+1:T}) \cdot p(w \mid x_{1:i-1}, x_{i+1:T}) \\ &= \log \sum_{w \in \mathcal{V}} \exp\{\log p(x_{1:i-1}, w, x_{i+1:T}) + \log p(w \mid x_{1:i-1}, x_{i+1:T})\} \\ &= \operatorname{LogSumExp}([\log p(x_{1:i-1}, w, x_{i+1:T}) + \log p(w \mid x_{1:i-1}, x_{i+1:T})]_{w \in \mathcal{V}})\end{aligned}$$

The term inside the "LogSumExp" function is the vector generated by computing the term for each $w \in \mathcal{V}$. Hopefully that can be done in parallel, and without major numerical issues. A good build-in implementation of "LogSumExp" should also be fine. Once this is computed, we can compute

$$\operatorname{logodds}(p_i(x)) = \log p_i(x) - \log(1 - \exp \log p_i(x))$$

Finally, the attribution score a is given by

$$a_i(x) = \operatorname{logodds}(p(x)) - \operatorname{logodds}(p_i(x))$$

Now, the last thing to check is if there is a clean way to compute $[\log p(x_{1:i-1}, w, x_{i+1:T})]_{w \in \mathcal{V}}$ and $[\log p(w \mid x_{1:i-1}, x_{i+1:T})]_{w \in \mathcal{V}}$. For the first term, we could in theory just evaluate the language model at all words in the vocabulary for all indices in the sequence. Because there are T scores, $|\mathcal{V}|$ words, and the model requires T calls to get the probability of a sequence, this would take

$O(|\mathcal{V}|T^2)$ which is really nasty because Wikitext is a character-level dataset. If $i = 1$, then every call $\log p(x_t \mid w, x_{2:t-1})$ will depend on the w and have to be re-computed. One fact that we can take advantage of is that, for example, $\log p(x_2 \mid x_1)$ is a repeated call for all attribution scores greater than $a_i(x)$ for $i > 2$. This can be cached. Is there any benefit to caching calls such as $\log p(x_t \mid w, x_{2:t-1})$? I don't believe they would be used again. It seems we cannot beat the $O(|\mathcal{V}|T^2)$, but we can save half of the calls which could be useful.

As for the $[\log p(w \mid x_{1:i-1}, x_{i+1:T})]_{w \in \mathcal{V}}$ term, this is essentially masking each word and computing the distribution on the value of the masked word. Not much insight here either, other than the fact that the sequence can be converted into a batch of T sequences, each with a word masked, and can be pushed through in $O(1)$ (assuming the work is well-scheduled on the GPU). This becomes a little difficult when a batch of sentences are given to score originally, but we will ignore this unless it becomes a problem.