Methods

- Novel, “reverse engineering” approach to finding text summaries (find a better word... in an information sense, not in a writing short summary sense). Given a target sequence and a given LLM, we want to find the input embedding to this LLM, which maximizes the likelihood for the LLM to generate the target sequence.
- Training can be done with simply forward passes through the LLM model

For a given Large Language Model, $\mathcal{M}$, we want to find the input embedding, $\boldsymbol{s}^*$, that maximizes the likelihood of generating a given target sequence, $\{t_i\}_{i=1}^{L}$, where L denotes the length of the target sequence and t_i represents token i.

To express that we generate tokens with the LLM, based on length E input-embedding $\boldsymbol{e}$, we use the notation $\mathcal{M}(\boldsymbol{e})$. To express the number of tokens, k that we generate, we use the subscript $\mathcal{M}_k(\boldsymbol{e})$. This function call returns an output vector of length V, where V denotes the vocabulary size. We denote the output vector as $\boldsymbol{o}$. Each element o_v of $\boldsymbol{o}$ represents the probability of generating token v as the next token. We predict the next token, by selecting the largest element of $\boldsymbol{o}$. Explicitly, $t_1 = \arg\max \mathcal{M}_1(\boldsymbol{e})$.

We denote the probability that $\mathcal{M}$ generates the target sequence, given the input embedding, as $p(t_1, \ldots, t_L|\boldsymbol{s})$.

We estimate these embeddings, by maximizing the conditional likelihood that this embedding generates the target sequence for a given LLM. We use a gradient-based optimization algorithm to maximize this likelihood.

These embeddings have the same dimension as the input embedding of $\mathcal{M}$, length E.

$$\boldsymbol{s}^* = \arg\max_{\boldsymbol{s}} \prod_{i=1}^{L} p(t_1, \ldots, t_L|\boldsymbol{s})$$

$$\boldsymbol{s}^* = \arg\max_{\boldsymbol{s}} \sum_{i=1}^{L} \log p(t_1, \ldots, t_L|\boldsymbol{s})$$

$$\boldsymbol{s}^* = \arg\max_{\boldsymbol{s}} \log p(t_1, |\boldsymbol{s}) + \log p(t_2, |\boldsymbol{s}, t_1) + \ldots + \log p(t_L, |\boldsymbol{s}, t_1, \ldots, t_{L-1})$$

$$\boldsymbol{s}^* = \arg\max_{\boldsymbol{s}} \log \mathcal{M}_1(\boldsymbol{s}) + \log \mathcal{M}_1([\boldsymbol{s}, \boldsymbol{e}_1]) + \ldots + \log \mathcal{M}_1([\boldsymbol{s}, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_{L-1}])$$

$$\boldsymbol{s}^* = \arg\max_{\boldsymbol{s}} \sum_{i=1}^{L} \log \mathcal{M}_L(\boldsymbol{s})$$

We can make this expression more explicit, by using the LLM in our notation. For example $p(t_1, |\boldsymbol{s}) = \mathcal{M}(\boldsymbol{s})$

In the training process, we execute the following algorithm. First, we initialize the input embedding, $\boldsymbol{s}$. Second, we generate a sequence of length L, with the LLM and the current input embedding.

```
Initialize s
i ← 0
while Convergence == False do
    i ← i + 1
    o^(i) ← M(s)
    l^(i) ← Likelihood(o^(i), {t_i}_1^L)
    s^(i+1) ← GradientDecent(s^(i), l^(i))
    Convergence ← CheckConvergence(s^(i+1), s^(i))
end while
```