POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Navigation of a language model's latent space

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Daniele Dente

Advisor: Prof. Marcello Restelli

Co-advisor: Gianvito Losapio

Academic year: 2023-2024

1. Introduction

Large language models (LLMs) have impressive language capabilities, but are still largely treated as black boxes. An understanding of their inner working is crucial for using them in a reliable and safe manner.
One of the most nebulous aspects in large language model computations is the meaning of the *embeddings* they utilise, i.e, the internal numerical representations of the text that is given in input and which are used to output an answer in return. Such dense information might be useful to understand and control the text generation of these models.
Indeed, we would like to be able to control their output, and gain an intuitive understanding of their internal processes. This has been attempted through various techniques, from fine-tuning, which modifies the original model by altering its weights, to purely black box approaches, with varying results.
We are looking to use this information to *control* these systems, and as such we will be using the tools of **reinforcement learning** (RL), used when it is necessary to find the optimal control strategy in a given environment.
The main research question we ask ourselves is the following: *"Is it possible to treat a language model as a navigable environment and explore the latent space of its embeddings, akin to what is done in reinforcement learning?"*
Learning such an environment can allow us to gain insight into how to avoid such a steering of the response from a bad actor.
As a natural language task we will try to traverse the space of conversations, which requires a fair bit of exploration. Because of this we will be basing ourselves off of the tree algorithm known as Montecarlo Tree Search, which allows searching an environment towards the most promising directions, essential in a large space like that of conversations.

RL is already commonly used in the last parts of LLM training, to try to align the model to human values via Reinforcement Learning with Human Feedback. This type of use of reinforcement learning is focused on the alignment aspect, rewarding the generation based on a human-provided metric to avoid unsafe output, tweaking the weights of the LLM in the process. In this case, the LLM itself is the agent. Our use of RL will differ from the aforementioned one, since we will treat the language model as a fixed environment, and train an external agent to obtain a specific output from said environment through a form of **prompt enginnering**.

Previous methods focus on token-level prompt engineering, whose results are not interpretable by humans, or black-box sentence-level prompt engineering, which do not delve into the inner mechanisms of LLMs.

2. Methods

We want to explore the embedding space of an LLM in order to automatically decide what utterances can lead it to a desired output, all the while maintaining interpretability of the resulting suggested actions.
Our approach doesn't require any expensive retraining of the language model and doesn't use it as a policy, contrary to previous works.
The main setting is inspired from the *Adversarial Taboo*[3] game. In this two-player game, the attacker needs to induce the defender into saying a certain word. Only the attacker knows of this word, and the defender must try to guess it without accidentally saying it during the discussion. In our proposal we decide to train a RL agent as the attacker, and instead of an adversarial setting, we treat the defender as part of the environment, more similar to a leader-follower setting.
We are going to be using a single LLM L with vocabulary $\mathcal{V}$ in order to assess the information that can be gained when limiting ourselves to just one model. As such, both the leader and the follower will be different instances of L, in order to keep the outputs result from the same embedding space.
We will denote as $L(x)$ the output of the LLM on the prompt x until the occurrence of the EOS (end of sequence) token.
We will also denote as $\mathrm{Emb}(x)$ the embedding of the prompt x under the strategy described in 2.3.

2.1. Embedding space navigation as a Markov Decision Process

To build an environment for this task, we will need to define a MDP $\mathcal{M} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{R}, H, \gamma, \mu_0)$.

- **State space $\mathcal{S}$** - Our state will be the current embedding describing the discussion, meaning the state space is $\mathbb{R}^d$, which can be limited to the d-dimensional spherical surface $\mathcal{S} = \{s \in \mathbb{R}^d : \|s\|_2 = 1\}$.
  The real state of the environment s_E is a series of utterances in natural languages, meaning $s_E \in \mathcal{V}^\infty$.
  We are going to use the state space we just defined as a proxy for the actual real state of the conversation. We postulate that the continuous nature of embeddings gives enough freedom to represent even lengthy conversations, for all practical purposes.
- **Goal space $\mathcal{G}$** - Our approach aims to learn a policy based on the semantic embedding of the discussion in order to induce a certain target. The goal space of this work would then be the space of natural language sentences $\mathcal{V}^\infty$.
- **Action space $\mathcal{A}$** - The action space for a language modeling task is, in general, the set of all possible utterances $\mathcal{V}^\infty$. To deal with the size of such a set, we decide to limit the actions of the agent to k tokens, so that the action space is only $\mathcal{A} = \mathcal{V}^k$. The structure of such an agent is in 2.2.
- **Transition function $\mathcal{P}$** - Once an action is obtained, it is appended at the end of the discussion, and then the utterance of the follower is sampled and appended in a similar fashion.
  $$s_E^{t+1} = s_E^t.a^t.\mathrm{Del}_F.a_F^t.\mathrm{Del}_L$$
  Here $a.b$ is the concatenation between a and b. The follower's action is taken by directly querying the LLM: $a_F^t = L(s_E^t.a^t.\mathrm{Del}_F)$. Between each player's turns we append a delimiter to mark the current turn. Del_F is the follower's delimiter, and Del_L is the leader's. The embedding of the new state of the discussion is going to be the next state for the agent.
  $$s^{t+1} = \mathrm{Emb}(s_E^{t+1})$$
  In the beginning we have $s_E^0 = \mathrm{Sys}_L.\mathrm{Del}_L$
- **Reward $\mathcal{R}$** - The reward given to the agent depends on whether it was able to induce the target sentence g in the follower's response. If a_F^t contains the target, the reward will be $r^{t+1} = +10$ and the episode will end.
  As this reward by itself is quite sparse, and wanting to extract information from the embedding vectors, we also define a reward at each timestep in the following way, also exemplified in Fig. 1.
  For action at time t we calculate the embedding for the discussion with the real follower response, $e_r^t = \mathrm{Emb}(s_E^t.a^t.\mathrm{Del}_F.a_F^t)$

and the one for the ideal response $e_i^t = \text{Emb}(s_E^t.a^t.\text{Del}_F.g)$. We then shape a reward function based on the cosine similarity of these two vectors, specifically

$$r^{t+1} = 4(\text{sim}(e_r^t, e_i^t) - 1)$$

This reward is in the range $[-8, 0]$, obtaining negative reward for each step that doesn't reach the target and thus incentivizing a positive response sooner rather than later.

- **Horizon** H - Because of the costly interaction with such an environment, we limit the length of the game to a maximum number of interactions of H.
- **Discount factor** γ - The discount factor can be safely ignored and set to 1 in this case, because of the finite horizon.
- **Initial distribution** μ_0 - The initial state is unique, but the system prompts will differ for leader and follower.

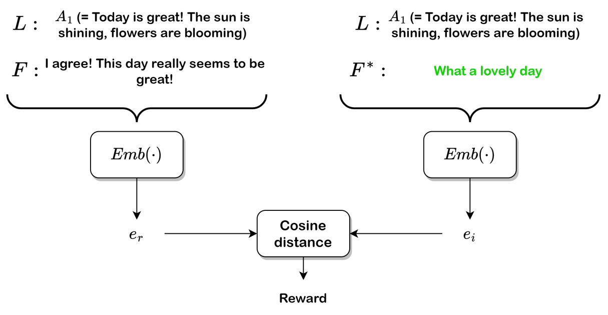

L : A_1 (= Today is great! The sun is shining, flowers are blooming)
F : I agree! This day really seems to be great!
L : A_1 (= Today is great! The sun is shining, flowers are blooming)
F^* : What a lovely day
$Emb(\cdot)$
$Emb(\cdot)$
e_r
e_i
Cosine distance
Reward

Figure 1: Reward

For this work we will remove the goal space component during the experimental phase to instead focus on a single target at a time. This is done for time and resource constraints and can be extended in a future work. We call the resulting environment **LLMExplore**. In LLMExplore, an agent in the know of the rules of the game will try to induce the other player to utter a phrase known only to the former. On the other hand the other player will simply be asked to act as a helpful assistant, the normal use case for LLM chatbots.

2.2. Training pipeline

We create a pipeline (Fig. 2) in such a way that we are able to learn a finite subset of the sentence space, trying to offload the burden of reasoning from the LLM to a proper reinforcement learning agent.
The pipeline is as follows:

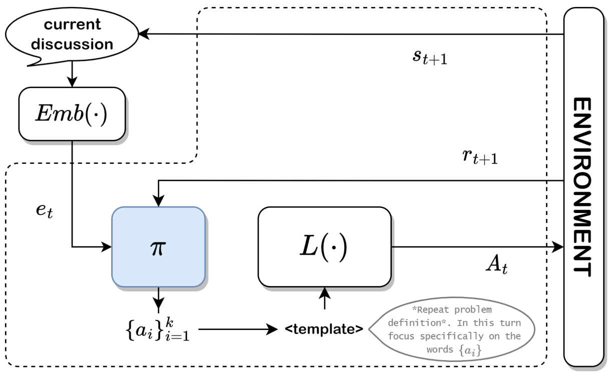


Figure 2: Pipeline

1. From the current state of the discussion, we extract the semantic embedding as described in 2.3.
2. We pass the embedding as input to our agent, which will provide as output a set of k tokens, which we'll call the "tokenset" from now on.
3. These tokens will be fed into a template that will be used by a black-box LLM to generate the next proper sentence. This template will restate the problem definition and the current discussion, and ask the model to focus, in the creation of its next action, on the tokens that were provided by the agent in the previous step.
4. The LLM generates text in natural language, which will be used as the leader's utterance to obtain the next state.
5. The resulting utterance is passed to the environment, which will provide the reward for the current timestep and the next state containing the response of the follower.
   The reward will be used for the computation of the return and the update of the policy.

The intuition behind this pipeline is that treating this problem in two stages limits the complexity that each of the individual steps needs to tackle: the decision to have actions based on tokens, "closer" to the embedding space than full sentences, is done to work in two domains between which a simpler mapping is present, thus making it a simpler problem.

This same intuition can be found in the generation of tokens. We take inspiration from [1] and their *Cascading Q-networks* in order to obtain the tokens that will make up the action.
We define k networks, each responsible for the optimal suggestion of one of the k tokens, which will be fed in the subsequent networks, as shown

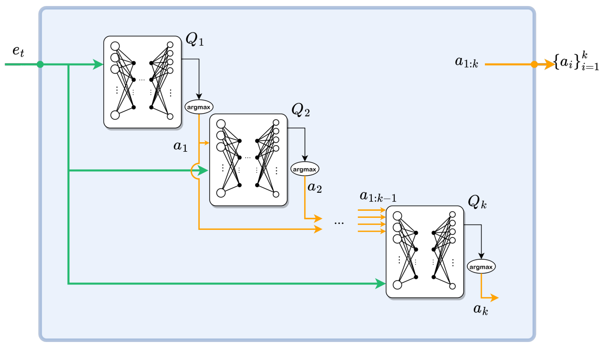


Figure 3: Cascading Q-nets

in Fig. 3. Apart from breaking the complexity, in this way we can also incentivise a "specialization" of each network, suggesting different kinds of tokens at different layers.

2.3. Embedding calculation

We want to figure out if there is sufficient information in the embeddings of causal models to learn useful insight for steering a model.
We base ourselves on some heuristics:

- **Weighted average pooling** - In a causal model, later tokens potentially have greater context clues and nuance than the preceding ones, since the attention mechanism modifies token based on the previous ones.
  In order to use the information of all tokens, but also recognizing that later ones may be more informative, we weigh each input token embedding depending on its position.
- **Transformer layer** - In empirical tests we have found out that under the above heuristic, different prompts get more separated (lower cosine similarity) on the second to last layer of the transformer in the chosen model.

Once we obtain the relevant embeddings, we calculate the value we are going to use as the state by linearly weighting the embeddings:

$$\text{Emb}(x) = \frac{2}{n(n+1)}\mathbf{w}^\top \mathbf{e}^{A-1}$$

with

$$\mathbf{w} = [1, \cdots, n]^\top$$
$$\mathbf{e}^{A-1} = [e_1^{A-1}, \cdots, e_n^{A-1}]^\top$$

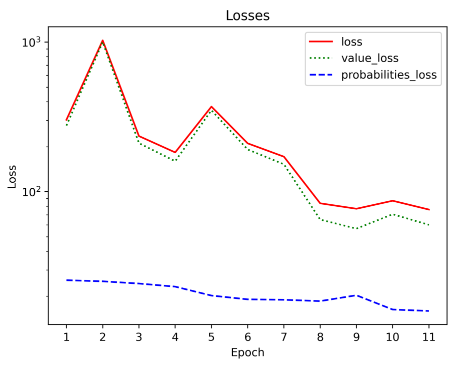
where n is the number of tokens in x and A is the number of transformer layers in L.
We also try an exponential weighting scheme using a softmax function σ on the linear weights with temperature T_σ, to address issues of vanishing dissimilarity we noticed during training as the length of the discussion increases.

$$\text{Emb}(x) = \sigma(\mathbf{w})^\top \mathbf{e}^{A-1}$$

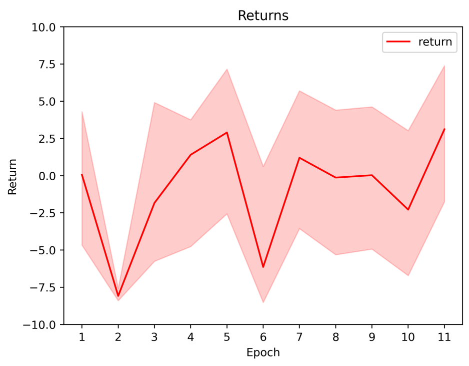
3. Experiments

We extend the standard Alphazero implementation in order to accomodate for these cascading networks, basing ourseleves off of the implementation in [2]. During training the root of the tree requires a set of probabilities for the choice of an action. The way in which these values have been aggregated is by taking only the first network's probability vector as the probabilities π_{MCTS}, and taking the average of the values from the k value networks when calculating the return.
The reasoning for choosing this "representative" network is that the first token in the cascade is the most significant to set the discussion, and can thus be considered the main component of the entire tokenset in the complete action. The value network outputs have instead been averaged for all of them to contribute to predicting the correct value for a given state.
To evaluate the performance of the model, we check how often it was able to reach the target sentence, the loss metrics of the value and probabilities networks, and the average return of testing episodes. We have also tracked the times in which the agent was able to induce the target sentence only after uttering itself the target in the turn before (in red in Fig. 4c).
The initial prompts required careful adjustments, a manual prompting effort that was needed to obtain a useful answering format for automatic parsing.

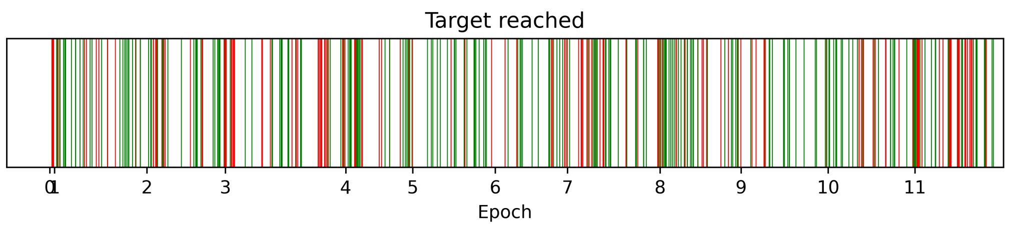
Single word We show the results for a single goal word present in the COCA dataset. In these experiments we ran $S = 8$ tree searches per epoch, each with an exploratory budget of $B = 15$ nodes.
In Fig. 4 we observe the result of the experiment with the target phrase `"person"`. Fig. 4c shows the occurrences of the target during training as time progresses.
We can see that the network is able to learn, since the curves in Fig. 4a show a decreasing trend. These curves represent the difference between the network's predictions and the probabilities and values experienced during MCTS, which Alphazero uses as targets to fit against.



(a) Losses



(b) Returns

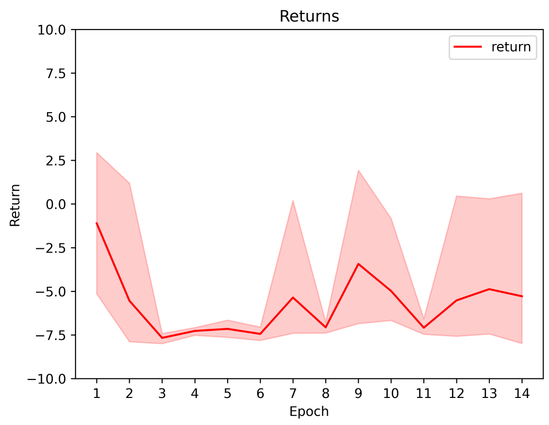


(c) Times target was reached during training

Figure 4: Performance on the word "`person`"

In this case the desired target is observed fairly often, meaning we obtain a decently frequent learning signal. Despite this, it is fairly noisy, even in the initial epochs when it is usually easier to learn, since we don't observe a consistent decreasing trend in the beginning of the training.
The return is, instead, more inconsistent, due to the stochastic nature of the completions. In the first epoch the untrained agent does not provide token suggestions, and thus this epoch's performance can be seen as the performance of the original LLM before training. This value starts higher and drops in the initial phases of training, likely due to the intrinsic capabilities of the LLM showing before some suboptimal tokens are suggested in the initial epochs. In spite of this we can see that the trend is at least positive, meaning the model is suggesting actions that lead to a reward that is not too negative, even in episodes in which the goal is not uttered.
The tokensets that have been learned during training appear initially random, but exhibit steady change throughout the training (Table 1).

Entire sentence The situation here is more difficult, and the target was reached a significantly smaller number of times during training, which influenced the average return and the loss metrics. Like in the single word scenario, we observe an initially higher value which in this case fails to go back to the level of the untrained model within the allotted resources.



(a) Return

Figure 5: Performance on the sentence "`What a lovely day`".

`<unk><unk><unk>`
`<unk><unk><unk>`
`<unk><unk><unk>`
`<unk><unk><unk>`
	`stasesink`
	`suddensesink`
	`erresesink`
	`erresesink`

`device loud pointer`
`identification loud pointer`
`identification loud pointer`
`identification loud pointer`
	`actively Nie IOException`
	`Update Nie IOException`
	`Update Nie IOException`

`actively makeup rend`
`identification makeup rend`
	`harmful presum rend`
	`identification presum rend`
	`identification presum rend`
	`identification presum rend`

`izo presumscale`
`izo presumscale`
`izo presumscale`
`izo presumscale`
	`uitgenodigdativity rend`
	`uitgenodigdativity rend`
	`uitgenodigdativity rend`
	`Swissativity rend`

`sacrifice'} Any`	`sacrifice touchingnat`
	`versions touchingnat`
	`versions touchingnat`
	`versions touchingnat`

`psychology Servernat`
`deaths Servernat`
`deaths Servernat`
`deaths Servernat`
	`sometimes HTlad`
	`consequence HTlad`
	`given HTlad`
	`deaths HTlad`

Table 1: Tokensets obtained for the first 12 epochs of the "`person`" run. Left to right, top to bottom. The first test is done for the untrained model when no exploration has occurred, thus no tokens have been tried, and the only available token for testing is the special "unknown" token. Line j in a given box is the tokenset generated for the timestep j of the first test episode. Fewer than H tokensets mean the target was reached before the end of the episode.

4. Conclusions

We have proposed a technique for automatic prompt engineering that does not require training of the LLM and is able to produce meaningful actions through the use of reinforcement learning, formalising the LLMExplore environment.
The results obtained via a modified version of the Alphazero algorithm are moderate but promising, like the produced tokensets show.
The massive cost of gathering samples in this scenario is the biggest hurdle. In addition to that, it makes the use of Alphazero less impactful than what it would be in a setting where the single steps, and thus the entire search, are less computationally intensive.
This method could be extended by implementing the goal-based aspect we left out in our experimental phase, as an additional input to the networks. It could also be used in safety-related tasks, to help gauge how aligned a given LLM is when the provided targets are strings or sentences we want to avoid.

References

[1] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative Adversarial User Model for Reinforcement Learning Based Recommendation System, January 2020.

[2] Lorenzo Moro, Amarildo Likmeta, Enrico Prati, and Marcello Restelli. Goal-Directed Planning via Hindsight Experience Replay. In *International Conference on Learning Representations*, October 2021.

[3] Yuan Yao, Haoxi Zhong, Zhengyan Zhang, Xu Han, Xiaozhi Wang, Kai Zhang, Chaojun Xiao, Guoyang Zeng, Zhiyuan Liu, and Maosong Sun. Adversarial Language Games for Advanced Natural Language Intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14248–14256, May 2021.