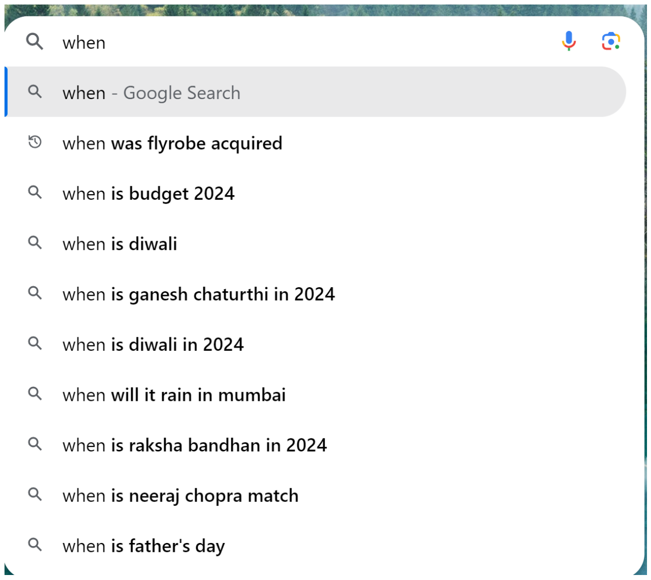
Application of Reinforcement Learning in NLP: Autocompletion

Pranjal Patel

August 2024

1 RL for Autocompletion

Autocompletion is a very important aspect of NLP, widely used in search engines, text editors like VS Code, and social media applications for messaging. It involves predicting what the user wants to search for based on what words he has already written.



when
when - Google Search
when was flyrobe acquired
when is budget 2024
when is diwali
when is ganesh chaturthi in 2024
when is diwali in 2024
when will it rain in mumbai
when is raksha bandhan in 2024
when is neeraj chopra match
when is father's day

Figure 1: Autocompletion

2 Nature of the Task

2.1 Language Variations

Each of us have unique searching styles like some prefer writing full text while others tend to write very short descriptions for searches. For example,

- "Who won 2 bronze medals from India in Olympics 2024?"
- "Indian to win 2 medals Olympics 2024"

2.2 Context

It is very essential to catch the context in which the user is searching something, or there can be real chaos if the agent suggests something which could hurt the sentiments of the user.

2.3 Real Time Changes

Adapting to real time changes in user patterns and improving upon the provided feedback to match the real time context of the user is a crucial task to achieve.

3 Why RL is a Good Fit

RL would work in this task since the task fits in the basics of RL, in the sense that based on the state of the environment, the agent would suggest a probable continuation and it would get a corresponding reward, positive if the user accepts any of the suggestions or writes the suggested himself/herself, and negative if (s)he writes something different than suggested.

3.1 Sequential Decision Making

RL's ability to consider discounted future rewards also into account is very useful.

3.2 Exploration and Exploitation

This provides flexibility since the user's patterns can change over time and the agent then would also need to explore.

4 The Environment

The environment consists of the user who is interacting with the agent.

5 State Space

A possible state would essentially include:

- Past n words that the user has typed
- Context (from the past words or from recent browsing history or data from other sister applications, like Google may use data from calendars, etc, (Maybe not due to Privacy restrictions))
- User patterns

Hence, the state space would include all combinations of all such possible states.

6 Action Space

Based on the current state, the agent can suggest the incomplete word or complete the phrase.
Action space would include all possible combinations of such actions.

7 Reward Function

- +1: If the user types chooses the suggested phrase/ if (s)he types the same
- -1: If the user types in something very different, which changes the context
- 0: If the suggested text did not change the context of what the user typed

8 Final Model

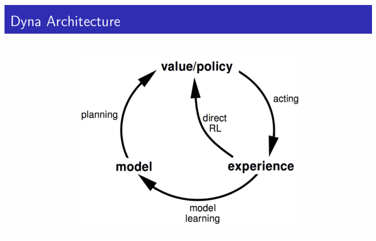
In my opinion, Dyna-Q would be the most suitable for this job.

First, the agent can learn how the user types and model it as an MDP. It can then train itself as much as it wants by sampling on the MDP. Along with this, every time a user also searches, it can improve upon its MDP and gradually learn to model the exact search patterns of the user.

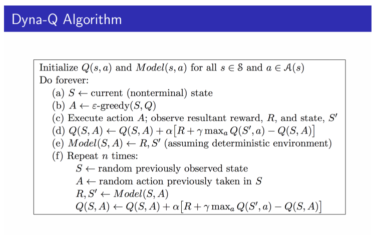
In this way, Dyna would consider 2 sources of experience:

- Real experience: Sampled from the environment (whenever user actually searches something on the engine)
- Simulated experience: Sampled from the model (approximate MDP)

Hence, it would be a great advantage over other methods.



(a) Dyna



Dyna-Q Algorithm

```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S,Q)$
    (c) Execute action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
```

(b) Dyna PseudoCode

Figure 2: The Dyna-Q algorithm

9 Example

User types in 2 words → The agent predicts based on the already existing MDP for the user, if user is new, predicts based on universal patterns → user accepts / rejects / types similar to the suggestion → corresponding reward → update MDP → user types another word → ...... When the user is not searching, agent can train itself on the latest MDP available.

10 Other Methods

Other methods which can be useful:

1. Using pre-trained models like GPTs can be helpful.
2. Seq2Seq models → kind of neural network architecture, which consist of an encoder and a decoder for the same.
3. N-gram models: predict stochastically the next word based on previous n-1 words.