ToT

20기 김민재

1. Abstract

이 논문에서는 Tree-of-Thought (ToT) 프레임워크를 소개하며, 이는 대형 언어 모델 (LLM) 의 문제 해결 능력을 향상시키기 위한 새로운 접근 방식이다. ToT는 인간이 문제를 해결할 때 시행착오를 거치며 다양한 해결 경로를 탐색하는 트리 기반 사고 방식에서 영감을 받았다.

ToT 프레임워크는 프롬프터 에이전트, 검증 모듈 (checker module), 메모리 모듈, ToT 컨트롤러 등의 추가 모듈을 사용하여 LLM과 다중 라운드 대화를 수행.

• 메모리 모듈: 문제 해결 과정의 상태와 기록을 저장하여 필요 시 되돌아갈 수 있도록 한다.

• ToT 컨트롤러: 해결 방향이 막히면 이전 단계로 돌아가 다른 경로를 탐색.

이 기법의 효과를 검증하기 위해 Sudoku 퍼즐을 해결하는 ToT 기반 시스템을 구현하였으며, 실험 결과 ToT 프레임워크가 Sudoku 해결 성공률을 상당히 증가시킴을 확인했다.

2. Introduction

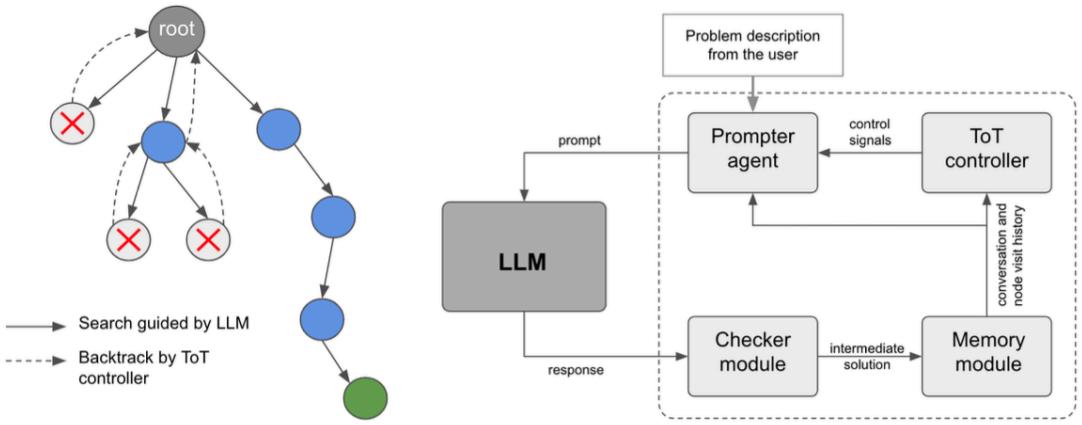
최근 GPT-4와 같은 자기회귀 기반 대형 언어 모델 (LLM) 은 다양한 논리적 및 수학적 문제 해결에서 뛰어난 성능을 보이고 있다.

하지만 긴 논리적 추론이 필요한 문제(Long-range reasoning tasks) 에서는 한계를 보인다.

LLM의 문제 해결 성능이 제한되는 주요 원인은 두 가지.

1. 정확성 검증 부족: 사람이 문제를 풀 때는 중간 과정에서 지속적으로 검증을 수행하지만, LLM은 자동으로 이를 수행하지 않음.
2. 선형적 응답 생성 방식: LLM은 이전 토큰을 기반으로 다음 토큰을 생성하기 때문에, 한 번 실수가 발생하면 수정하기 어렵다.

이를 극복하기 위해, 인간이 문제를 해결할 때 사용하는 트리 기반 탐색 방법을 LLM에 적용한 Tree-of-Thought (ToT) 프레임워크를 제안.



(a) ToT search strategy.

(b) ToT software system.

3. Related Works

ToT는 기존 연구들에서 발견된 LLM의 문제 해결 능력을 확장하려는 시도 중 하나이다.

• Self-Taught Reasoner (STaR): 잘못된 답변을 제거하고 올바른 답변을 훈련하는 방식이지만, 자동화하기 어려움이 있다.

• Chain-of-Thought (CoT): 단계별 추론을 유도하는 방식이지만, 사람이 직접 prompt를 설계해야 하는 단점.

• AutoGPT, ReAct, Reflexion: 추가 모듈을 활용하여 LLM의 능력을 확장하지만, 기존 방식은 명시적 백트래킹 기능이 부족.

ToT는 이러한 기존 접근 방식과 차별화되는 명확한 Backtracking 기능을 도입하여 더 넓은 해결 공간을 탐색할 수 있도록 합니다.

4. Architecture

4.1 Tree-of-Thought 프레임워크 개요

ToT는 LLM을 트리 탐색의 휴리스틱 도구로 활용하여 단계별 문제 해결을 수행하는 방식.

• LLM은 short-range reasoning을 수행.

• 검증 모듈과 컨트롤러가 이를 보완하여 long-range reasoning을 개선.

4.2 ToT 모듈

1. Checker Module
    - 생성된 부분 해답이 올바른지 검증

- Sudoku의 경우, 간단한 규칙 기반 검증 시스템으로 구현

2. Memory Module
   - 이전 대화 및 해결 과정을 저장
   - 되돌아가거나 다른 방향을 탐색할 수 있도록 지원
3. ToT Controller
   - 해결 과정의 진행 상황을 모니터링
   - ToT 컨트롤러는 크게 두 가지 주요 규칙에 따라 작동한다.
     1. 잘못된 해결 경로를 되돌리는 백트래킹 기능
        - LLM이 생성한 중간 답변이 checker module에 의해 잘못된 것으로 판정되면, ToT 컨트롤러는 해당 경로를 무효화하고 이전 상태로 되돌아가(Backtrack) 다른 해결 경로를 탐색하도록 지시한다.
        - 예를 들어, Sudoku 퍼즐을 해결하는 경우, 숫자가 겹치는 규칙 위반이 발생하면 ToT 컨트롤러는 이전 단계로 돌아가 다른 숫자를 시도하도록 한다.
     2. 해결 경로가 막히면 다른 방향을 탐색
        - 해결 과정이 진행되더라도, 일정 횟수 이상 같은 경로에서 답을 찾지 못할 경우, ToT 컨트롤러는 현재 경로를 포기하고 새로운 방향으로 탐색하도록 유도한다.
        - 예를 들어, LLM이 Sudoku 퍼즐에서 특정 부분을 반복적으로 시도하지만 진행이 없으면, 컨트롤러는 새로운 숫자 배치를 시도하도록 프롬프트를 수정할 수 있다.
   - 더 고도화된 ToT 컨트롤러는 학습기반(policy network 방식)

     신경망 입력 데이터:
     - 최근 탐색한 노드 정보: ToT 탐색 트리에서 최근 방문한 노드들의 정보
     - 현재 노드의 검증 결과: 현재 노드가 검증 모듈에서 유효한지 여부
     - 이전 백트래킹 이력: 최근 몇 번의 백트래킹 기록

     신경망 출력:
     - 다음 행동 결정
       1. 현재 경로에서 계속 진행할지
       2. 부모 노드로 돌아갈지
       3. 상위 몇 단계까지 백트래킹할지
4. Prompter Agent
   - LLM이 더 효과적으로 답을 생성하도록 프롬프트 조정

- LLM이 생성한 답변을 기반으로 추가 힌트를 제공

프롬프터 에이전트는 문제 해결을 위해 적절한 프롬프트를 생성하는 역할을 한다.

기본적으로 문제 설명 + 현재까지의 해결 과정 + 다음 단계 요청의 형태로 프롬프트를 구성한다.

4.3 ToT 시스템 학습 (Training the ToT System)

• Policy Gradient Reinforcement Learning, REINFORCE을 활용하여 ToT 컨트롤러와 프롬프터를 학습

• 올바른 답을 생성할 경우 +1, 실패할 경우 -1의 보상을 부여하여 강화 학습 수행

4.4 ToT 기반 문제 해결 과정

1. 사용자가 문제를 입력
2. 프롬프터가 LLM에 초기 프롬프트를 제공
3. LLM이 응답을 생성 → 검증 모듈이 확인
4. 정답이 나오지 않으면, ToT 컨트롤러가 방향을 조정
5. 반복하여 해결 시도 (K번까지 가능)

5. Evaluation & Experiments

ToT 프레임워크의 성능을 평가하기 위해 Sudoku 퍼즐 해결 실험을 진행.

5.1 ToT 기반 Sudoku 퍼즐 해결기

- Python으로 구현
- “gpt-3.5-turbo” 모델 사용
- 백트래킹 기능이 있는 ToT 컨트롤러 적용

5.2 실험 결과

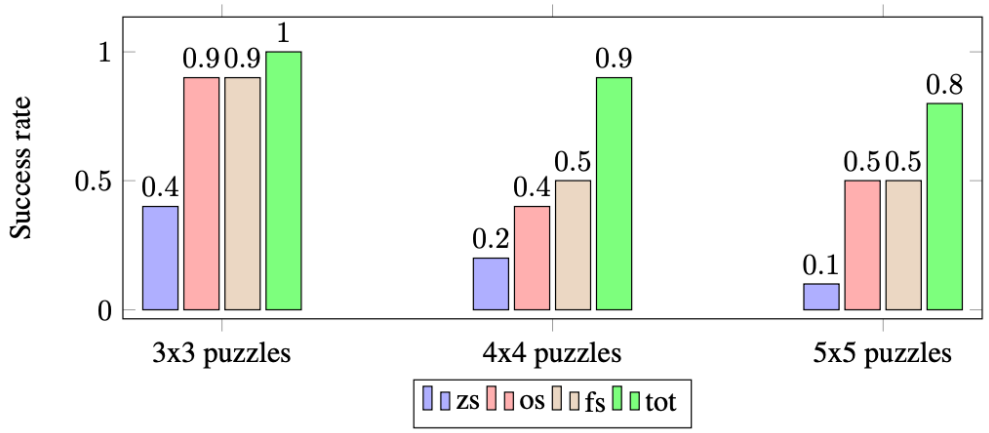
비교 대상

1. Zero-shot solver (zs): Sudoku를 직접 해결하도록 요청
2. One-shot solver (os): 단일 예제 포함

3. Few-shot solver (fs): 여러 예제 포함
4. ToT 기반 Solver (tot)

성공률 비교 (퍼즐 크기: 3x3, 4×4, 5×5)

- Zero-shot: 최저 성공률
- One-shot & Few-shot: 3×3 퍼즐에서는 효과적이지만, 큰 퍼즐에서는 성능 저하
- ToT Solver: 가장 높은 성공률 (특히, 5x5 퍼즐에서 60% 높은 성공률)



6. Discussion & Future Work

ToT 프레임워크는 LLM의 한계를 극복하기 위해 추가 모듈을 결합한 접근 방식으로, 문제 해결 성능을 크게 향상시시킨다.

향후 연구 방향

1. 더 일반적인 문제 해결에 적용: Sudoku뿐만 아니라 다양한 논리 및 수학 문제로 확장
2. 신경망 기반 검증 모듈 적용: 보다 복잡한 문제에서도 적용 가능하도록 개선
3. 강화 학습을 활용한 최적화: ToT 컨트롤러를 더 정교한 방식으로 개선
4. Self-play 학습 도입: 스스로 문제를 생성하고 해결하는 방식으로 성능 향상 기대