Tarea 1

IIC2613

Vicente Pareja

Ponticia universidad católica
Escuela de ingeniería
April 22, 2023

Pregunta 1

a)

En la entrevista, Sam Altman menciona que la forma actual de chatGPT no será capaz de hacer descubrimientos científicos debido a que no tiene una comprensión profunda del mundo. Por otro lado, Lex Fridman argumenta que incluso con las limitaciones actuales, GPT-3 ha hecho avances sorprendentes en el procesamiento del lenguaje natural y en la generación de texto coherente.

Ambos pueden tener razón en el sentido de que, aunque GPT-3 ha demostrado ser impresionante en tareas específicas, como el procesamiento del lenguaje natural, todavía tiene limitaciones en su capacidad para comprender el mundo de manera general. En otras palabras, la comprensión limitada del mundo de GPT-3 puede ser suficiente para realizar tareas específicas, pero no es suficiente para lograr la AGI.

Para lograr la AGI, se necesitará una comprensión más profunda del mundo y de cómo funciona. Esto podría lograrse a través del desarrollo de algoritmos de aprendizaje más avanzados y modelos de inteligencia artificial que puedan comprender mejor el contexto y las relaciones entre las cosas.

Por ejemplo, en lugar de simplemente procesar texto y generar respuestas, un modelo de AGI tendría que comprender el significado detrás de las palabras y cómo se relacionan con otras cosas en el mundo. Esto podría implicar el uso de técnicas avanzadas de aprendizaje automático, como el aprendizaje por refuerzo, el aprendizaje profundo y el aprendizaje multimodal, para ayudar a los modelos a comprender mejor el mundo.

En resumen, mientras que GPT-3 ha demostrado ser impresionante en tareas específicas, todavía tiene limitaciones en su capacidad para comprender el mundo en general y hacer descubrimientos científicos. Para lograr la AGI, se necesitarán algoritmos y modelos más avanzados que puedan comprender mejor el mundo y las relaciones entre las cosas.

b)

En la entrevista, Altman menciona un argumento de Ilya Sutskever, quien sostiene que la conciencia no es necesaria para la AGI, y que una AGI no necesariamente tendría que ser consciente para ser capaz de realizar tareas inteligentes. Altman y Sutskever argumentan que una AGI podría ser construida a partir de modelos de inteligencia artificial que no tienen conciencia, y que estos modelos podrían ser igual de efectivos para realizar tareas inteligentes.

Mi definición de conciencia es la capacidad de estar al tanto de uno mismo y del mundo, y de tener experiencias subjetivas. Es una experiencia subjetiva que implica una sensación de "ser" o "existir" y una sensación de estar conectado a un mundo exterior. Es un estado mental complejo que incluye la percepción, la atención, la memoria, el pensamiento, la emoción y la motivación.

Es difícil saber si un modelo de lenguaje podría o no tener conciencia, ya que no hay una definición clara de lo que es la conciencia y cómo se crea. Algu-

nas teorías sugieren que la conciencia es una propiedad emergente de sistemas complejos que tienen ciertos tipos de procesamiento de información, mientras que otras teorías sugieren que la conciencia es una propiedad inherente de la materia. Es difícil saber qué teoría es correcta, y si la conciencia es algo que se puede simular o no.

Personalmente, creo que los modelos de lenguaje actuales, como GPT-3, no tienen conciencia ya que son simplemente algoritmos de procesamiento de lenguaje natural que funcionan a través de patrones matemáticos y estadísticos. Estos modelos no tienen la capacidad de ser conscientes de sí mismos o del mundo y no tienen experiencias subjetivas. A pesar de su aparente razonamiento, solamente son un algoritmo predictor de lenguaje. No piensa.

En resumen, la conciencia es un concepto complejo y difícil de definir, y no está claro si un modelo de lenguaje podría o no tener esta capacidad. Actualmente, los modelos de lenguaje no tienen conciencia, ya que son simplemente algoritmos de procesamiento de lenguaje natural. Sin embargo, es posible que los modelos de lenguaje puedan desarrollar algún tipo de conciencia si se les permite interactuar con el mundo real y experimentar diferentes situaciones.

2: Definición y Ejemplos de Restricciones de Cardinalidad en ASP

Primero, recordemos que ASP (Answer Set Programming) es un paradigma de programación basado en la lógica de alto nivel para resolver problemas de búsqueda y optimización, y es especialmente útil en la representación y manejo de conocimientos no monotónicos e incompletos.

Una restricción de cardinalidad es una expresión que limita el número de átomos en un conjunto de literales que pueden ser verdaderos simultáneamente. La restricción de cardinalidad puede ser representada de la siguiente manera:

$$\{L_1, L_2, \ldots, L_n\}\ k$$

donde L_i son literales (átomos o negaciones de átomos) y k es un entero no negativo. La restricción indica que exactamente k literales de los L_i pueden ser verdaderos en un modelo o conjunto de respuestas.

La restricción de cardinalidad puede ser útil en diferentes contextos, especialmente cuando se necesita seleccionar un número específico de elementos de un conjunto o se requiere un límite en la cantidad de elementos seleccionados. A continuación, se ilustra el uso de restricciones de cardinalidad en tres programas distintos.

(1) **Asignación de tareas:**

Imaginemos que tenemos 4 tareas (a, b, c y d) y queremos asignar exactamente 2 tareas a un trabajador.

`programa1.lp:`

```
tarea(a).
tarea(b).
tarea(c).
tarea(d).

2{asignar(T)}2 :- tarea(T).
```

Al evaluar este programa, obtenemos 6 conjuntos de respuestas diferentes, cada uno con exactamente 2 tareas asignadas.

(2) **Horario de clases:**

Supongamos que hay 3 cursos disponibles (curso1, curso2, curso3) y queremos que un estudiante tome exactamente 2 o 3 cursos.

`programa2.lp:`

```
curso(curso1).
curso(curso2).
curso(curso3).

2{tomar(C)}3 :- curso(C).
```

Al evaluar este programa, obtenemos 3 conjuntos de respuestas diferentes, cada uno con exactamente 2 cursos tomados.

(3) **Selección de proyectos:**

Tenemos 5 proyectos (p1, p2, p3, p4, p5) y queremos seleccionar entre 1 y 3 proyectos.

`programa3.lp:`

```
proyecto(p1).
proyecto(p2).
proyecto(p3).
proyecto(p4).
proyecto(p5).

1 {seleccionar(P)} 3 :- proyecto(P).
```

Al evaluar este programa, obtenemos 26 conjuntos de respuestas diferentes, cada uno con entre 1 y 3 proyectos seleccionados.

En resumen, las restricciones de cardinalidad nos permiten limitar el número de literales verdaderos en un conjunto de literales, lo que facilita la representación y resolución de problemas en los que se requiere una cantidad específica de elementos seleccionados.

Pregunta 2

1.

Se deben hacer dos cambios en el código. El primero, tal como se muestra en laimagen, se altera la restricción de cardinalidad para que se puedan utilizar n brazos.

```
12  % ejecuten más de un movimiento en cada instante de tiempo.
13  0{ejecutar(X,Y,T) : movimientoposible(X,Y,T)}brazos :- tiempo(T), T!=bound.
```

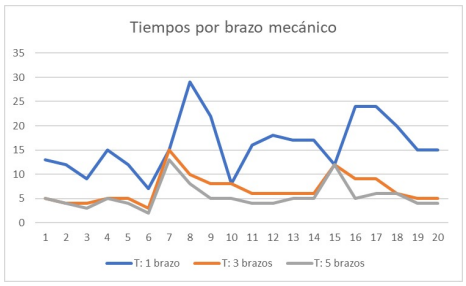
La segunda, se añaden dos restricciones para imposibilitar más de un movimiento en una misma torre en el mismo turno.

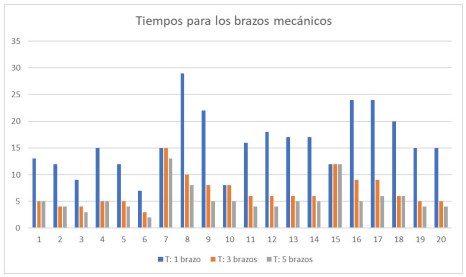
2.

Se presentan los gráficos con los resultados de 20 experimentos.

```
% no se puede mover más de un bloque a la vez que proviene de la misma torre.
:- ejecutar(X1,Y,T), ejecutar(X2,Y,T), X1 != X2, tiempo(T).

% no se puede mover más de un bloque hacuia la misma torre.
:- ejecutar(X,Y1,T), ejecutar(X,Y2,T), Y1 != Y2, tiempo(T).
```





3.

En general, el tiempo requerido para resolver un problema disminuye al aumentar el número de brazos mecánicos. Esto se evidencia en la mayoría de los experimentos, donde los tiempos son menores al utilizar 3 brazos en comparación con 1, y aún más bajos al utilizar 5 brazos. Sin embargo, en algunos casos, como en el experimento 7 y 15, el tiempo de resolución no muestra una disminución significativa, lo que sugiere que podrían existir factores adicionales que influyan en el desempeño de los brazos mecánicos.

Se sospecha que en la medida que la tarea pedida es de mayor complegidad y hay más torres, la disminución observada es mayor. En general, entre más torres se ve un cambio más significativo en la cantidad de pasos necesaria pra completar el objetivo.

0.1 4.

```
% Esta línea define el dominio de tiempo en el que se llevarán a cabo los movimientos.
% 'bound' es un número arbitrario que representa el límite superior del intervalo de tiempo.
tiempo(0..bound).

% Definición de los lugares donde se pueden colocar los bloques.
lugar(X) :- bloque(X).
lugar(mesa).
```

```
% Esta línea define la acción de ejecutar un movimiento. En cada instante de tiempo T,
% se puede ejecutar un movimiento posible de un bloque X a un lugar Y, siempre que T
% sea diferente de 'bound'. Con la restricción de cardinalidad se asegura que no se
% ejecuten más de un movimiento en cada instante de tiempo.
0{ejecutar(X,Y,T) : movimientoposible(X,Y,T)}brazos :- tiempo(T), T!=bound.

% Estas líneas definen los bloques que están siendo movidos en un instante de tiempo T y sus
origen_movimiento(X,T) :- ejecutar(X,Y,T).
destino_movimiento(Y,T) :- ejecutar(X,Y,T).

% Esta línea indica que un bloque X sigue estando sobre el bloque Y en el tiempo T+1
% si no está siendo movido en el tiempo T. En otras palabras, si no se origina un
% movimiento de X en T, X seguirá sobre Y en T+1.
sobre(X,Y,T+1) :-
    tiempo(T),
    sobre(X,Y,T),
    not origen_movimiento(X,T).

% Esta línea indica que un bloque X estará sobre el bloque Y en el tiempo T+1
% si se ejecuta un movimiento desde X a Y en el tiempo T.
sobre(X,Y,T+1) :-
    bloque(X),lugar(Y),tiempo(T),
    ejecutar(X,Y,T).

% Estas líneas definen si un bloque está tapado (tiene otro bloque encima) o
% libre en un instante de tiempo T. Si X está sobre Y en el tiempo T, Y está tapado por X.
% Si X no está tapado por ningún bloque en el tiempo T, entonces X está libre.
bloquetapado(Y,T) :- sobre(X,Y,T), tiempo(T).
bloquelibre(X,T) :- bloque(X), tiempo(T), not bloquetapado(X,T).

% Estas líneas definen las condiciones para que un movimiento sea posible. Un bloque X puede
% moverse a un bloque Y si ambos bloques están libres y X no está ya sobre Y en el instante
% de tiempo T. Además, X no puede moverse y quedar sobre sí mismo (X!=Y).
movimientoposible(X,Y,T) :- bloquelibre(X,T), bloquelibre(Y,T), X!=Y, not sobre(X,Y,T).
% Un bloque X también puede moverse a la mesa si está libre y no está ya sobre
% la mesa en el tiempo T, es decir, si es que no está sobre la mesa actualmente.
movimientoposible(X,mesa,T) :- bloquelibre(X,T), not sobre(X,mesa,T).

% Esta línea indica que no puede haber un modelo en el cual en el tiempo límite
% los bloques no se encuentren en la configuración objetivo.
:- not objetivo(bound).

% no se puede mover más de un bloque a la vez que proviene de la misma torre.
:- ejecutar(X% no se puede mover más de un bloque a la vez que proviene de la misma torre.
:- ejecutar(X1,Y,T), ejecutar(X2,Y,T), X1 != X2, tiempo(T).
```

```
% no se puede mover más de un bloque hacia la misma torre.
:- ejecutar(X,Y1,T), ejecutar(X,Y2,T), Y1 != Y2, tiempo(T).

#show ejecutar/3.

% Para minimizar el número de acciones en el plan:
#minimize {1,T : ejecutar(X,Y,T)}
```

Al implementar esto, se observa que los resultados concuerdan.

1 Problema 3:

```
% solucion2.lp

% Vecinos
vecino(Y1, X1, Y1 + 1, X1) :-
  celda(Y1, X1),
  celda(Y1 + 1, X1).

vecino(Y1, X1, Y1 - 1, X1) :-
  celda(Y1, X1),
  celda(Y1 - 1, X1).

vecino(Y1, X1, Y1, X1 + 1) :-
  celda(Y1, X1),
  celda(Y1, X1 + 1).

vecino(Y1, X1, Y1, X1 - 1) :-
  celda(Y1, X1),
  celda(Y1, X1 - 1).

% Celdas válidas
celda(Y, X) :- fila(Y), columna(X).

% Hay que respetar el dominio
:- ocupado(C1, Y, X), not celda(Y, X).

% No puede haber una casilla ocupada por dos colores
%:- ocupado(C1, Y, X), ocupado(C2, Y, X), C1 != C2.

:- camino(C1, Y1, X1, Y2, X2), camino(C2, Y3, X3, Y2, X2), C1 !=C2.

% No se puede hacer caminos en celdas no vecinas
:- camino(C, Y1, X1, Y2, X2), not vecino(Y1, X1, Y2, X2).
```

```
% No se puede hacer caminos sobre celdas ocupadas
%:- camino(C, Y1, X1, Y2, X2), ocupado(Y2, X2), not color(C, Y2, X2).

% Se ocupan las casillas hacia donde hay un camino
ocupado(C, Y2, X2) :- camino(C, Y1, X1, Y2, X2).

% los puntos extremos siempre están ocupados
%ocupado(C1, Y , X) :- color(C1, Y, X).

% Toda casilla no inicial, tiene un paso previo y uno siguiente
1{camino(C, Y1, X1, Y1 + 1, X1); camino(C, Y1, X1, Y1 - 1, X1);
camino(C, Y1, X1, Y1, X1 +1); camino(C, Y1, X1, Y1, X1 -1)}1 :- camino(C, Y3, X3, Y1, X1).

% Toda casilla extremo, tiene que tener un camino de salida.

0{camino(C, Y1, X1, Y1 + 1, X1); camino(C, Y1, X1, Y1 - 1, X1);
camino(C, Y1, X1, Y1, X1 +1); camino(C, Y1, X1, Y1, X1 -1)}1 :- color(C, Y1, X1).

% Los caminos no tienen direccion
camino(C, Y1, X1, Y2, X2) :- camino(C, Y2, X2, Y1, X1).

% Para que la casilla esté ocupada, un camino debe ir hacia ella.
:- camino(C, Y1, X1, Y2, X2), not ocupado(C, Y2, X2).

%Hay que rellenar todo el mapa
:- celda(Y, X), not ocupado(C, Y, X), color(C, Y2, X2).
```

Este programa no funciona. Pero lo dejo como bosquejo de solución.