Understanding Machine Solving Of Complex Problems (Alpha Geometry)

Presented by
Aditya ,
Sanjiban,
Uday

INSPIRATION

Recently Google has developed AlphaGeometry an AI system that solves complex geometry Problems at a level comparable to that of a human Gold Medalist.

.

Moreover it is a great achievement in the field of machine learning/deep learning which has enhances in solving complex problems of ai using a combination of neural networks and under the rule bounding deduction engine.

It also become important due to its various applications such as medical, robotics in real world scenarios.

Project vision and mission

01. Understanding working of State Of The Art problem solving models

02. Analyzing the processes and methods involved in building a problem solving model

03. Building a small scaled version of problem solving model

Working of Problem Solving Models

Problem solving models such as Alpha Geometry function as a combination of two separate frameworks. One is the use of a large language model. This is trained on a large dataset containing a variety of problems from math olympiads. Due to this training the Large language model is able to predict approaches to large varieties of problems with high accuracy. The second framework consists of an inference engine. This engine helps us to check truth of propositions by combining true propositions to check whether the given proposition is satisfied or not. Thus with synchronisation of the two frameworks one is able to get an efficient problem solving model.

Analyzing the processes and methods involved in building a problem solving model

The processes involved in the inference engine are based upon the basic definitions. There also exists common definitions such as midpoints ,segments etc. Beams are also used which help to decide how far into an approach the algorithm goes. The beam search also helps in fast decoding from a trained model which has been used by alpha Geometry. In the graph file algebra has been defined and also includes basic geometrical shapes which help us to decode the geometrical problems. In the problem file we convert a problem statement from a string and define it in the language define in the. Theorem premisis has also been handled whichhelps us to keep track of the theorems. THese all functionalities have led to successfull implementation of problem solving model.

Ideation process

01

recognizing the problem with traditional methods and its limitations in 3d shapes and spatial reasoning

02

Before moving to a problems alphageometry has been trained with large amount of different problems which further helps in understanding and optimizing modern test problems.

03

Developing advanced algorithms: creating new mathematical models and algorithms for manipulating geometric data in 3d and higher dimensional spaces.

04

Real-World Applications: Applying these innovations in fields like robotics, 3D design, and computer vision to solve practical, complex problems.

Code Snippet

Code Link

```python
import math
from shapely.geometry import Polygon
import matplotlib.pyplot as plt
import geopandas as gpd

class Triangle:
    def __init__(self):
        self.sides = [0, 0, 0]  #for non type error
        self.angles = [None, None, None]
        self.area = None
        self.perimeter = None
        self.altitudes = [None, None, None]
        self.inradius = None
        self.circumradius = None

    def is_triangle_by_angles(self, angle_A: float, angle_B: float, angle_C: float) -> b
        return abs(angle_A + angle_B + angle_C - 180) < 1e-6 and all(angle > 0 for angle

    def is_triangle(self, a: float, b: float, c: float) -> bool:
        return a + b > c and a + c > b and b + c > a

    def calculate_sas(self, a: float, b: float, angle_C: float):
        if a <= 0 or b <= 0 or angle_C <= 0 or angle_C >= 180:
            raise ValueError("Sides must be positive and angle must be between 0 and 180

        angle_C_rad = math.radians(angle_C)
        c_squared = a**2 + b**2 - 2 * a * b * math.cos(angle_C_rad)

        if c_squared <= 0:
            raise ValueError("Invalid triangle dimensions.")

        self.sides = [a, b, math.sqrt(c_squared)]

        if not self.is_triangle(*self.sides):
            raise ValueError("The calculated sides do not satisfy the triangle inequalit

        self.angles[2] = angle_C
        self.angles[0] = math.degrees(math.asin((a * math.sin(angle_C_rad)) / self.sides
        self.angles[1] = 180 - self.angles[0] - self.angles[2]

        if not self.is_triangle_by_angles(*self.angles):
            raise ValueError("The calculated angles do not satisfy the angle sum propert

    def calculate_asa_or_aas(self, angle_A: float, angle_B: float, side_C: float):
        if angle_A <= 0 or angle_B <= 0 or side_C <= 0 or angle_A + angle_B >= 180:
            raise ValueError("Angles must be positive and their sum must be less than 18

        # Using the Law of Sines to calculate the other sides
        angle_A_rad = math.radians(angle_A)
```

```python
    raise ValueError("The calculated sides do not satisfy the triangle inequality.")

calculate_properties(self):
if any(side == 0 for side in self.sides):
    raise ValueError("Sides must be calculated before computing properties.")

# Calculate perimeter
self.perimeter = sum(self.sides)

# Calculate area using Heron's formula
s = self.perimeter / 2  # Semi-perimeter
self.area = math.sqrt(s * (s - self.sides[0]) * (s - self.sides[1]) * (s - self.sides[2]))

# Calculate altitudes
self.altitudes[0] = (2 * self.area) / self.sides[0]
self.altitudes[1] = (2 * self.area) / self.sides[1]
self.altitudes[2] = (2 * self.area) / self.sides[2]

# Calculate inradius and circumradius
self.inradius = self.area / s
self.circumradius = (self.sides[0] * self.sides[1] * self.sides[2]) / (4 * self.area)

display_properties(self):
print("Triangle Properties:")
print(f"Sides: a = {self.sides[0]:.2f}, b = {self.sides[1]:.2f}, c = {self.sides[2]:.2f}")
print(f"Angles: A = {self.angles[0]:.2f}°, B = {self.angles[1]:.2f}°, C = {self.angles[2]
print(f"Perimeter: {self.perimeter:.2f}")
print(f"Area: {self.area:.2f}")
print("Altitudes:")
print(f"  Altitude from vertex A to side a: {self.altitudes[0]:.2f}")
print(f"  Altitude from vertex B to side b: {self.altitudes[1]:.2f}")
print(f"  Altitude from vertex C to side c: {self.altitudes[2]:.2f}")
print(f"Inradius: {self.inradius:.2f}")
print(f"Circumradius: {self.circumradius:.2f}")

display_results(self):
self.display_properties()

# Triangle plotting (visualization)
angle_C_rad = math.radians(self.angles[2])
polygon1 = Polygon([(0, 0), (self.sides[0], 0),
                    (self.sides[1] * math.cos(angle_C_rad),
                     self.sides[1] * math.sin(angle_C_rad))])

p = gpd.GeoSeries(polygon1)
p.plot()
plt.title("Triangle Visualization")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.axis("equal")
```

Final reflections and future steps

The model we created is still very lacking and needs to be better. With future expansion we could possibly create a small version of an inference engine. This project has helped us gain knowledge about the vast field of AI and how AI can help us solving complex problems in mathematics. Overall our journey in learning this has been enriching and hope that we could continue this further.

Acknowledgement

We would really like to thank Dr Pradip Sasmal Sir for mentoring us in this project. Without his help we wouldn’t have been able to reach the level of understanding of the problem statement that we have now. We are truly grateful to him.

Thank you very much!

www.reallygreatsite.com