Reinforcement Learning for Multiple Addition

Mathematical Reasoning Project

May 18, 2025

Abstract

This report presents an implementation of a Reinforcement Learning (RL) agent designed to learn and execute multiple addition operations. The agent learns to transform expressions like $a + b + c$ into sequences of binary additions using parentheses, following the principle that addition is associative. We demonstrate the agent's capability by solving the addition problem $2 + 3 + 4$ and visualizing the solution path as a knowledge graph in Neo4j. This implementation showcases how RL can be applied to mathematical reasoning tasks involving multiple operands.

1 Introduction

Multiple addition can be systematically broken down into sequences of binary additions using parentheses. The key principles are:

- Addition is binary - we can only add two numbers at a time
- For n numbers, we need $(n - 1)$ additions
- Parentheses specify operation order
- Different groupings yield same result (associativity)

Our implementation uses Reinforcement Learning to teach an agent to apply these principles correctly.

2 Reinforcement Learning Components

2.1 Key RL Concepts

The implementation uses the same core RL concepts as the single addition case:

- **Agent**: The decision-maker that learns from experience
- **Environment**: The world with which the agent interacts
- **State**: The current mathematical expression

- **Action**: A decision about how to group or calculate
- **Reward**: Feedback signal indicating the quality of an action
- **Policy**: The agent's strategy for selecting actions
- **Value Function**: Estimation of future rewards from a state
- **Q-Function**: Estimation of future rewards from a state-action pair

2.2 Q-Learning Algorithm

The implementation uses Q-learning with the same update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

where:

- $Q(s,a)$ is the value of taking action a in state s
- α is the learning rate
- r is the reward received
- γ is the discount factor for future rewards
- s' is the next state
- $\max_{a'} Q(s',a')$ is the maximum Q-value for the next state

3 Implementation Components

3.1 Environment

The environment (`MultipleAdditionEnv`) represents the multiple addition problem and tracks:

- Current mathematical expression state
- Target result
- Available actions
- Step count and termination conditions

3.2 State Representation

States are represented by the `MultipleAdditionState` class, which contains:

- The current mathematical expression (e.g., "2 + 3 + 4", "(2 + 3) + 4")
- Step count for tracking progress

3.3 Actions

The agent can perform the following actions:

- `group_left`: Group from left to right (e.g., "2 + 3 + 4" to "(2 + 3) + 4")
- `group_right`: Group from right to left (e.g., "2 + 3 + 4" to "2 + (3 + 4)")
- `calculate`: Calculate the result of a simple addition
- `finish`: Conclude the calculation

3.4 Reward Structure

The reward structure guides the agent toward correct solutions:

- +1.0 for correct grouping steps
- +1.0 for correct calculations
- +5.0 for reaching the correct final result
- -1.0 for invalid actions
- -5.0 for finishing with an incorrect result

4 Example: Solving 2 + 3 + 4

For the addition problem 2 + 3 + 4, the agent can find multiple valid solution paths:

4.1 Left-to-Right Grouping

$$\begin{aligned} 2+3+4 &= (2+3)+4 && (2)\\ &= 5+4 && (3)\\ &= 9 && (4) \end{aligned}$$

4.2 Right-to-Left Grouping

$$\begin{aligned} 2+3+4 &= 2+(3+4) && (5)\\ &= 2+7 && (6)\\ &= 9 && (7) \end{aligned}$$

Both paths are valid and demonstrate the associativity of addition.

5 Knowledge Graph Representation

The solution is represented as a knowledge graph in Neo4j:

- **Nodes**: Mathematical expressions at each step
- **Edges**: Actions taken to transform expressions
- **Properties**: Step information, action types, and rewards

This graph provides a visual representation of the reasoning process, showing how the agent groups the numbers and performs the calculations.

6 Conclusion

This implementation demonstrates how Reinforcement Learning can be applied to multiple addition problems. The agent successfully learns to group numbers and perform calculations in a way that respects the associativity of addition. The knowledge graph representation provides insight into the reasoning process and can be used for educational purposes.

Future work could include:

- Extending the approach to more complex mathematical expressions
- Implementing other mathematical properties (commutativity, distributivity)
- Improving the agent's learning efficiency
- Developing a user interface for interactive exploration of the solution paths