Reinforcement Learning for Addition via Recursion

Mathematical Reasoning Project

May 10, 2025

Abstract

This report presents an implementation of a Reinforcement Learning (RL) agent designed to learn and execute the recursive definition of addition. The agent learns to transform expressions like $a+b$ into $(a+(b-1))+1$ recursively until reaching the base case $a+0=a$. We demonstrate the agent's capability by solving the addition problem $5+8$ and visualizing the solution path as a knowledge graph in Neo4j. This implementation showcases how RL can be applied to mathematical reasoning tasks.

1 Introduction

Addition is typically taught as a basic operation, but it can be formally defined using recursion:

$$a+b=\begin{cases} a & \text{if } b=0 \quad \text{(Base Case)} \\ (a+(b-1))+1 & \text{if } b>0 \quad \text{(Recursive Case)} \end{cases} \tag{1}$$

This recursive definition provides a systematic way to reduce complex addition problems to simpler ones. Our implementation uses Reinforcement Learning to teach an agent to apply this recursive definition correctly.

2 Reinforcement Learning Components

2.1 Key RL Concepts

Reinforcement Learning is a machine learning paradigm where an agent learns to make decisions by interacting with an environment. The key components are:

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

- **Action**: A decision made by the agent
- **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

Our implementation uses Q-learning, a model-free RL algorithm that learns the value of an action in a particular state. The Q-function is updated using:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)] \tag{2}$$

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 (`AdditionRecursionEnv`) represents the 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 `MathExpressionState` class, which contains:

- The current mathematical expression (e.g., "5 + 8", "(5 + 7) + 1")
- Step count for tracking progress

3.3 Actions

The agent can perform the following actions:

- `decompose`: Transform "a + b" to "(a + (b-1)) + 1"
- `further_decompose`: Further decompose a nested expression
- `apply_base_case`: Apply the rule (a + 0) = a
- `increment`: Perform an increment operation (e.g., (3 + 1) to 4)
- `calculate`: Calculate the final result
- `finish`: Conclude the calculation

3.4 Reward Structure

The reward structure guides the agent toward correct solutions:

- +1.0 for correct decomposition steps
- +1.0 for correctly applying the base case
- +1.0 for correct increment operations
- +5.0 for reaching the correct final result
- -1.0 for invalid actions
- -5.0 for finishing with an incorrect result

3.5 Agent

The `QLearningAgent` implements:

- Q-table for storing state-action values
- Epsilon-greedy policy for action selection
- Q-learning update rule
- Exploration rate decay
- Action history tracking

3.6 Knowledge Graph Builder

The `KnowledgeGraphBuilder` constructs a graph representation of the solution:

- Nodes represent mathematical expressions (states)
- Edges represent transformations (actions) between states
- The graph is exported as triplets for Neo4j storage

4 Training Process

The training process involves:

1. Initializing the environment with the problem "5 + 8"
2. Training the agent for a specified number of episodes
3. For each episode:
   - Reset the environment
   - Agent selects actions using epsilon-greedy policy
   - Environment provides rewards and next states
   - Agent updates Q-values using the Q-learning update rule
4. Tracking the best solution found during training
5. Building a knowledge graph from the best solution
6. Storing the graph in Neo4j

5 Example: Solving 5 + 8

For the addition problem 5 + 8, the ideal solution path is:

$$
\begin{aligned}
5+8 &= (5+7)+1 && (3)\\
&= ((5+6)+1)+1 && (4)\\
&= (((5+5)+1)+1)+1 && (5)\\
&= ((((5+4)+1)+1)+1)+1 && (6)\\
&= (((((5+3)+1)+1)+1)+1)+1 && (7)\\
&= ((((((5+2)+1)+1)+1)+1)+1)+1 && (8)\\
&= (((((((5+1)+1)+1)+1)+1)+1)+1)+1 && (9)\\
&= ((((((((5+0)+1)+1)+1)+1)+1)+1)+1)+1 && (10)\\
&= (((((((5+1)+1)+1)+1)+1)+1)+1)+1 && (11)\\
&= ((((((6+1)+1)+1)+1)+1)+1)+1 && (12)\\
&= (((((7+1)+1)+1)+1)+1)+1 && (13)\\
&= ((((8+1)+1)+1)+1)+1 && (14)\\
&= (((9+1)+1)+1)+1 && (15)\\
&= ((10+1)+1)+1 && (16)\\
&= (11+1)+1 && (17)\\
&= 12+1 && (18)\\
&= 13 && (19)
\end{aligned}
$$

6 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 decomposes the problem and builds up to the solution.

7 Conclusion

This implementation demonstrates how Reinforcement Learning can be applied to mathematical reasoning tasks. The agent successfully learns to apply the recursive definition of addition and solve problems like 5 + 8. 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 other mathematical operations (subtraction, multiplication)
- Implementing more complex mathematical reasoning tasks
- Improving the agent's learning efficiency
- Developing a user interface for interactive exploration of the solution paths