Reinforcement Learning for Arithmetic Circuit Simplification

Hanson Mo

February 4, 2025

Abstract

This report summarizes our work on training a reinforcement learning (RL) agent to simplify arithmetic circuits that compute polynomials. We present three sets of experiments:

1. **Constant Parameters:** Using fixed polynomial parameters ($n = 3$, $d = 3$).
2. **Dynamic Parameters:** Varying n and d over fixed ranges while keeping a constant structure.
3. **Randomized Domain:** Randomizing n, d, and the coefficients every episode.

We detail our environment design, RL methodology, experimental setup, and results.

1 Introduction

The goal of this project is to develop an RL agent that, given a polynomial, can construct an efficient arithmetic circuit to compute the polynomial. An arithmetic circuit is modeled as a collection of basic operations (addition, multiplication, and scalar multiplication) arranged in a directed acyclic graph. Our primary objective is to train the agent—using Proximal Policy Optimization (PPO)—to simplify the circuit, thereby reducing its overall "cost" (a proxy for complexity).

Polynomials are represented as:

$$(n, d, [c_1, c_2, \ldots, c_m])$$

where n is the number of variables, d is the degree, and the coefficient list $[c_1, \ldots, c_m]$ is arranged in lexicographical order. For homogeneous polynomials, the number of monomials is given by:

$$m = \binom{n+d-1}{d}.$$

2 Methodology

2.1 Environment Design

We developed a custom Gymnasium environment with two variants:

- **Constant Environment:** The polynomial parameters are fixed (e.g., $n = 3$ and $d = 3$). The circuit is initialized with:

$$\text{num\_add} = m - 1, \quad \text{num\_mul} = m, \quad \text{num\_scalar} = 0.$$

- **Dynamic/Random Environment:** The environment samples n, d, and the coefficients from predefined ranges for each episode. For example, n is sampled from $[2, 6]$, d from $[2, 6]$, and coefficients from 1 to 9. The number of monomials is computed as above, and the circuit is initialized similarly.

In both cases, the observation returned is a vector:

$$[\text{num\_add}, \text{num\_mul}, \text{num\_scalar}, \text{cost}, m],$$

where the cost is defined as:

$$\text{cost} = \text{num\_add} + \text{num\_mul} + \text{num\_scalar}.$$

The environment supports four discrete actions:

1. Combine addition gates.
2. Combine multiplication gates.
3. Factor out a scalar.
4. No operation.

2.2 Reinforcement Learning Setup

We employed the Proximal Policy Optimization (PPO) algorithm using StableBaselines3. In the more challenging experiments, we used a custom policy network with two hidden layers of 256 neurons each for both the policy and value networks. Key hyperparameters include:

- Learning rate: 0.0003
- $n\_steps = 2048$
- Batch size: 64
- Total timesteps: ∼200,000

3 Experimental Results

3.1 Constant Parameters Experiment

Setup:

- Fixed $n = 3$, $d = 3$ resulting in $m = \binom{5}{3} = 10$ monomials.
- Simple network architecture.
- Episodes typically ended in a fraction of the allowed steps.

Results:

- **Success Rate:** Nearly 100% of episodes reached an ideal circuit (cost = 0).
- **Average Reward:** Cumulative rewards consistently reflected full circuit simplification.

3.2 Dynamic Parameters Experiment

Setup:

- n and d were varied over fixed ranges, with the coefficient list remaining constant in structure.
- Increased maximum steps (e.g., 1000) allowed for more complex circuits.

Results:

- The agent learned to generalize over a range of polynomials.
- Performance was slightly lower than in the constant case, but success rates remained high.

3.3 Randomized Domain Experiment

Setup:

- n was sampled from $[2, 6]$, d from $[2, 6]$, and coefficients from 1 to 9.
- Maximum steps increased to 1000.
- A deeper network was used with two hidden layers of 256 neurons each.
- Evaluation was performed over 500 episodes.

Training Results (Sample Log):

Metric	Value
Rollout ep_len_mean	120
Rollout ep_rew_mean	118
FPS	915
Total timesteps	200704
Approx KL divergence	0.0169
Clip fraction	0.253
Entropy loss	-0.65
Explained variance	1
Learning rate	0.0003
Total loss	0.783
Policy gradient loss	-0.0159
Value loss	0.404

Evaluation Results:

- **Average Cumulative Reward:** 74.20 over 500 episodes.
- **Success Rate:** 90.8% of episodes reached an ideal circuit (cost = 0).

3.4 Interpretation of Results

- **Constant Parameters:** The agent quickly learned to simplify circuits when the polynomial structure was fixed. High success rates (close to 100%) and consistent rewards indicate that the task was relatively simple.
- **Dynamic Parameters:** With n and d varied over fixed ranges (but not completely randomized), the agent generalized well, though slight variability in performance was observed.
- **Randomized Domain:** When every episode presents a new, randomly generated polynomial, the task becomes significantly harder. The average episode length of 120 steps (well below the maximum of 1000) indicates that the agent often succeeds quickly, but the success rate of 90.8% (versus 100% in the simpler cases) shows that some instances remain challenging. The deeper network and increased training timesteps allowed the agent to achieve robust performance even under this variability.

4 Conclusion

This project demonstrates that reinforcement learning can be applied to the problem of arithmetic circuit simplification. We developed a custom environment that initially used fixed polynomial parameters and later extended it to a randomized domain. Our experiments show:

- With fixed parameters, the RL agent achieved nearly perfect performance.
- As the problem was made more challenging by randomizing n, d, and coefficients, the agent still learned an effective general policy, achieving an average cumulative reward of 74.20 and a success rate of 90.8% over 500 episodes.