Reinforcement Learning Techniques for Solving Optimization Problems

Jayotsana Sharma and Ryosuke Mori

École Centrale Nantes

June 24, 2025

Outline

Introdution

- **Motivation:** To address the non-differentiability and non-convexity of the objective function during optimization.
- **Objective:** To explore and validate the feasibility of Reinforcement Learning for continuous optimization.
- **Problem Statement:** To find the global optimum of a given objective function.

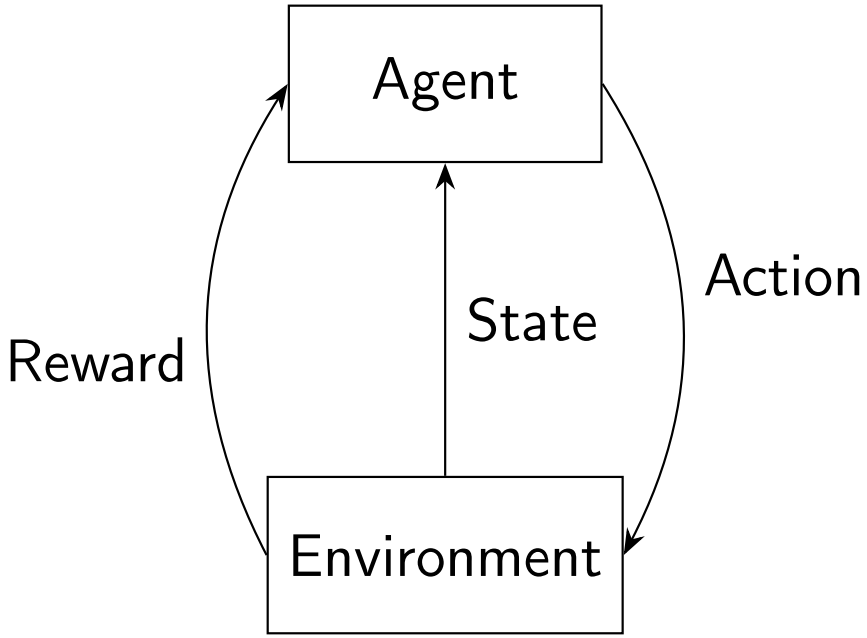
$$\max_x f(x)$$

or

$$\min_x f(x)$$

Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning in which an agent learns to make decisions by interacting with an environment in order to maximize some notion of cumulative reward.



Reinforcement Learning

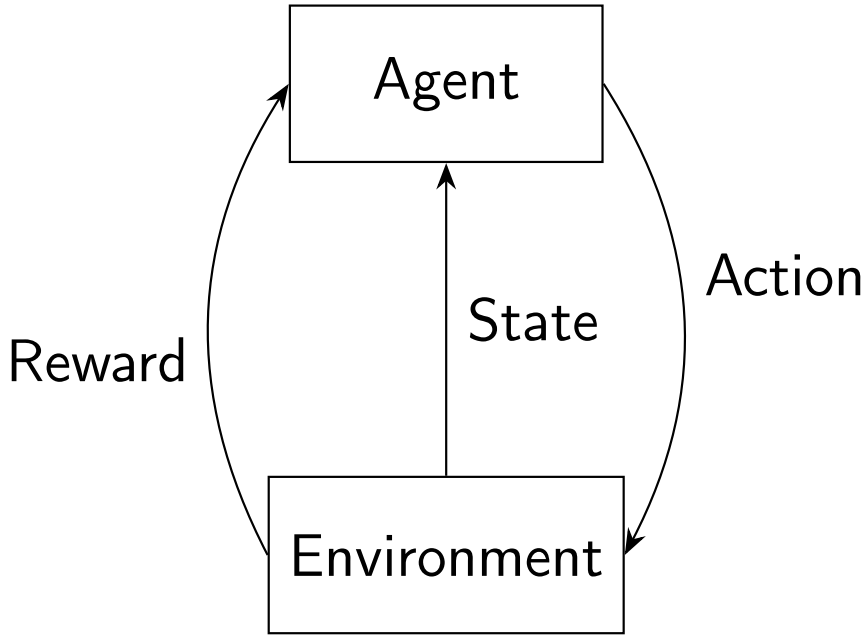


Fig.1: Agent–Environment Interaction

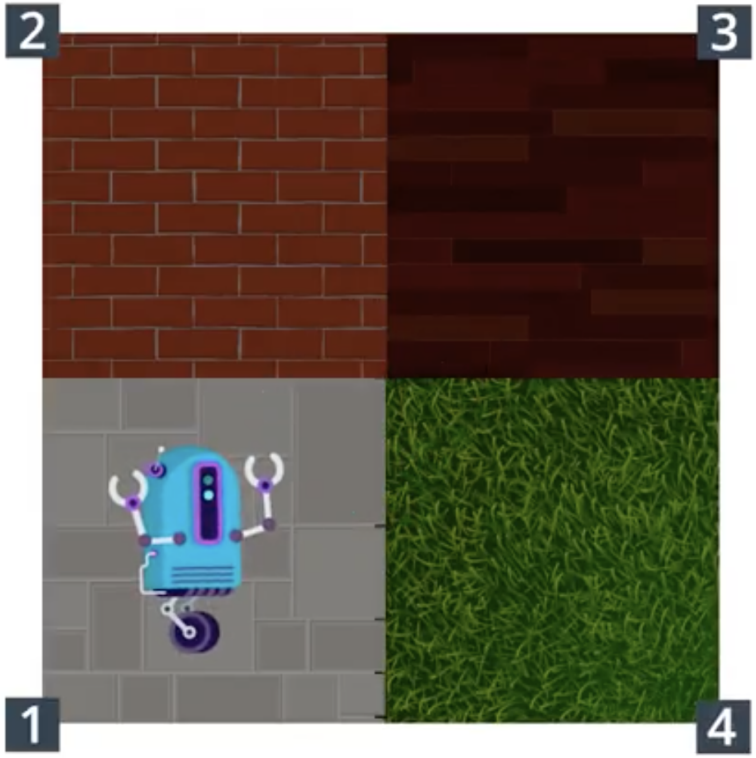


Fig.2: 2×2 Grid-World States & Actions

RL Approaches for Agent Learning

- **Value-Based Methods:** Discrete, small action spaces
  - Learn an action-value function $Q(s, a)$.
  - Examples: Q-Learning, Deep Q-Networks (DQN).
- **Policy-Based Methods:** Continuous or high-dimensional actions
  - Learn a stochastic policy directly:

$$\pi_\theta(a \mid s)$$

  - Popular Algorithms: REINFORCE, PPO, A2C, SAC, etc.
- **Our Choice: RL-OPT**
  - A variant of REINFORCE optimized for global search.
  - Well-suited to continuous, black-box objectives.

Methodology

Rosenbrock function:

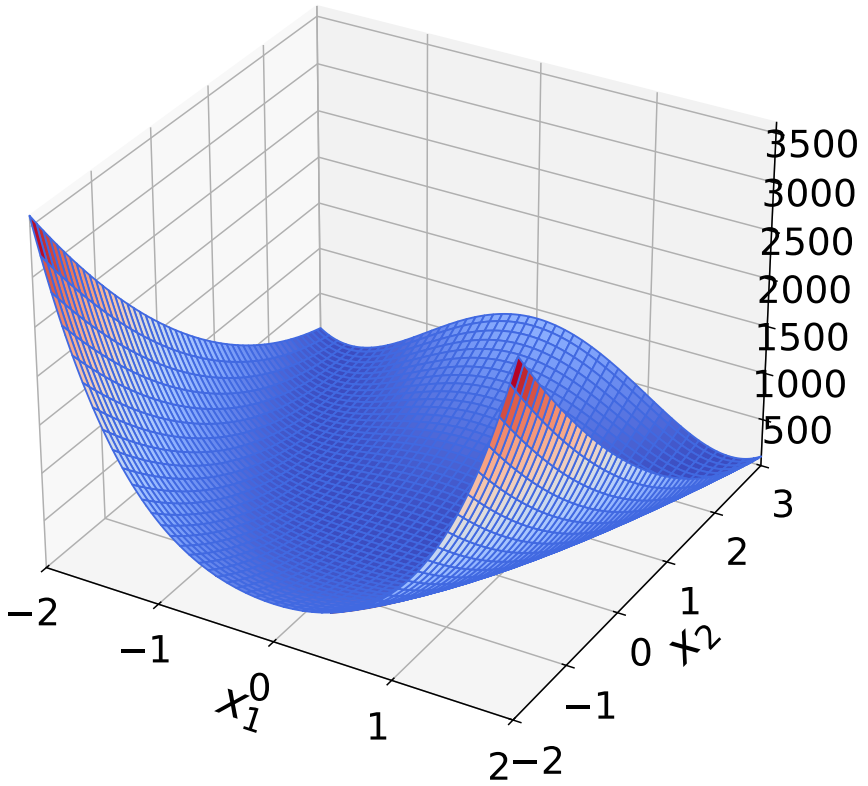


Figure 1: Objective Landscape

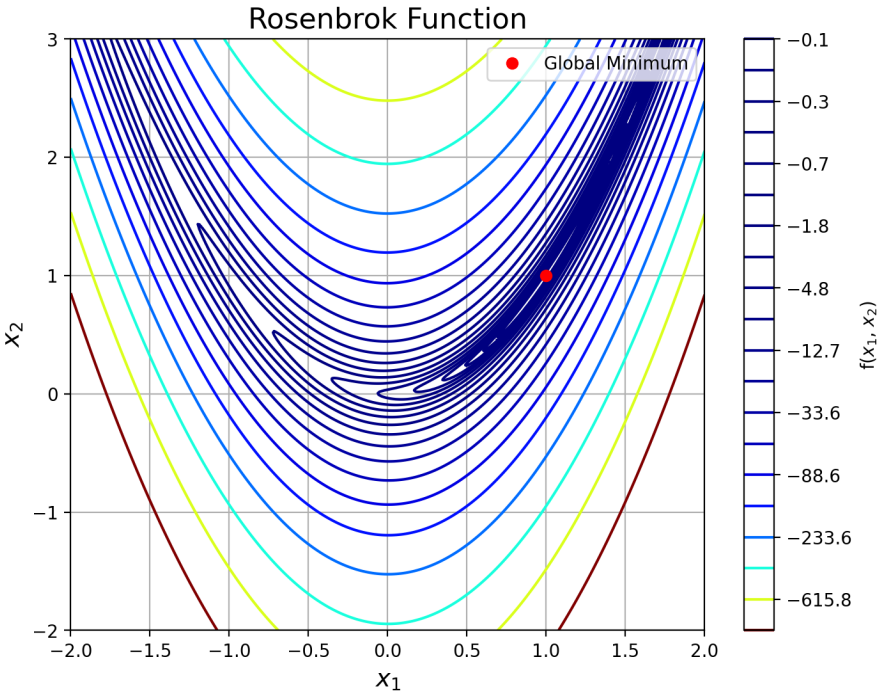


Figure 2: Convergence Contours

Methodology

Reinforce-Opt algorithm:

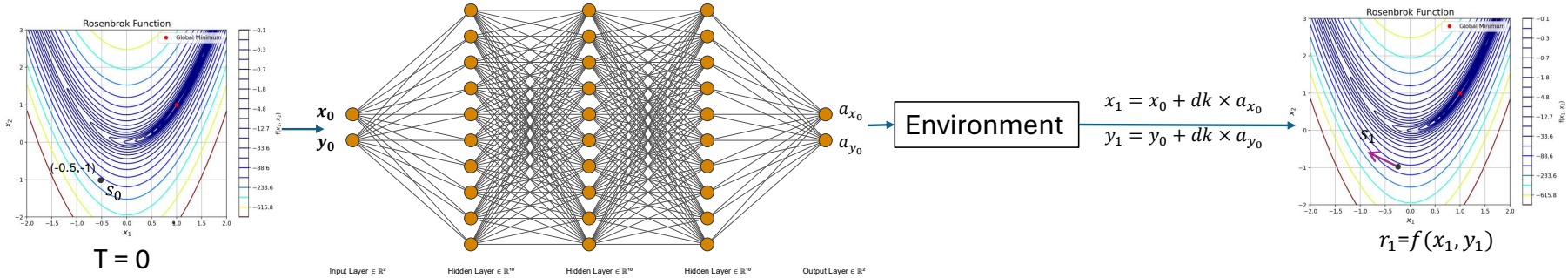


Figure: Generation of a single step

Methodology

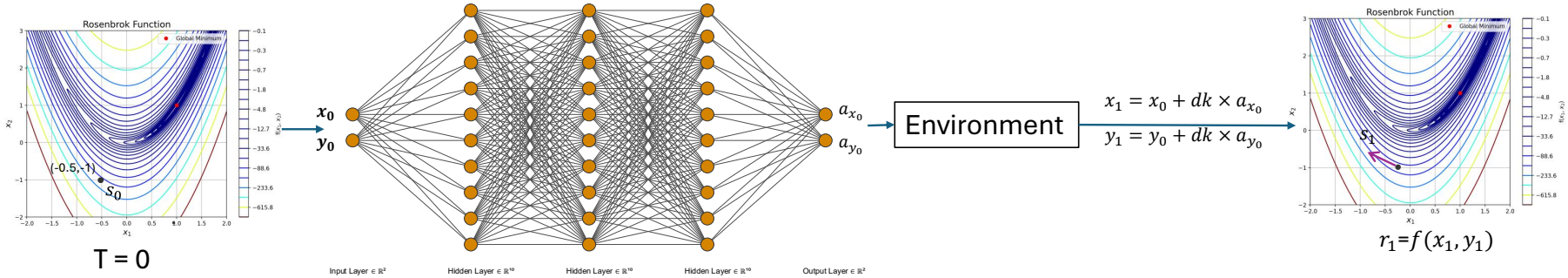


Figure: Generation of a single step

The action space per dimension is: $= \{-1, 0, 1\}$
Example action combinations when $n = 2$

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Methodology

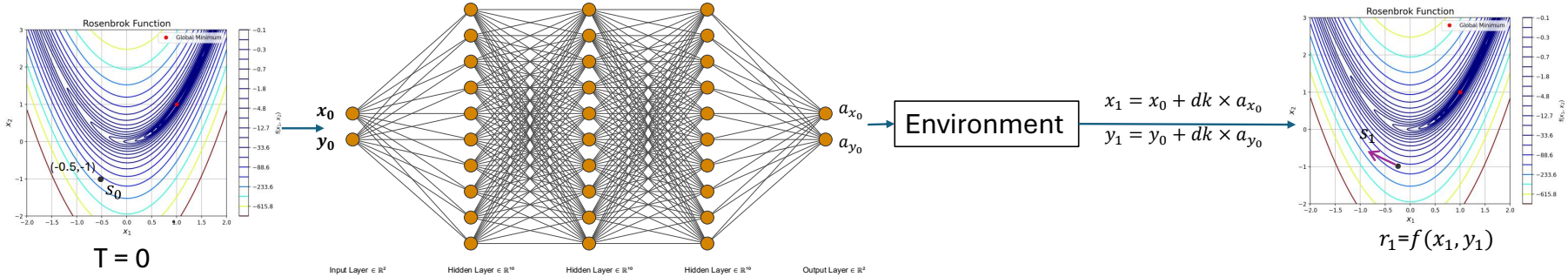


Figure: Generation of a single step

Methodology

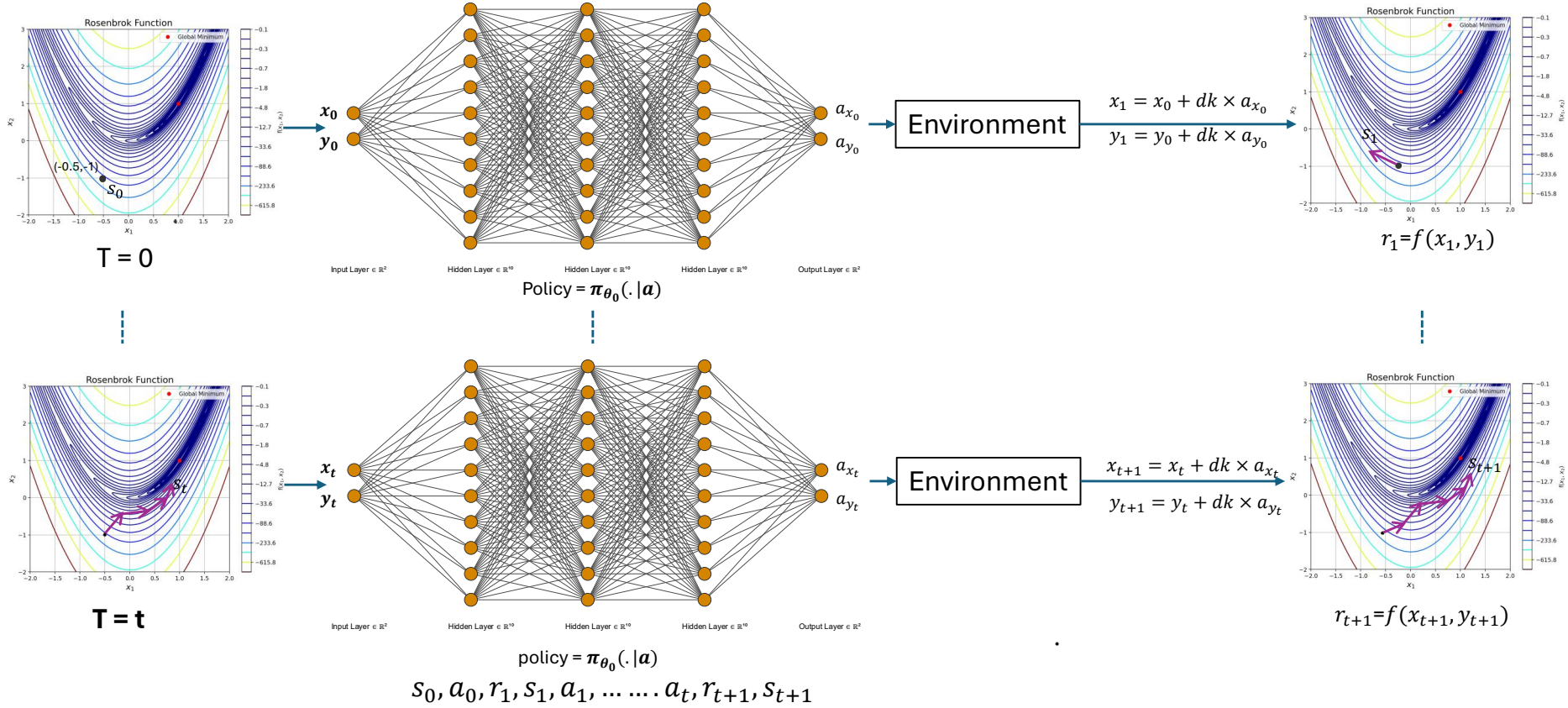


Figure: Generation of a single trajectory

Methodology

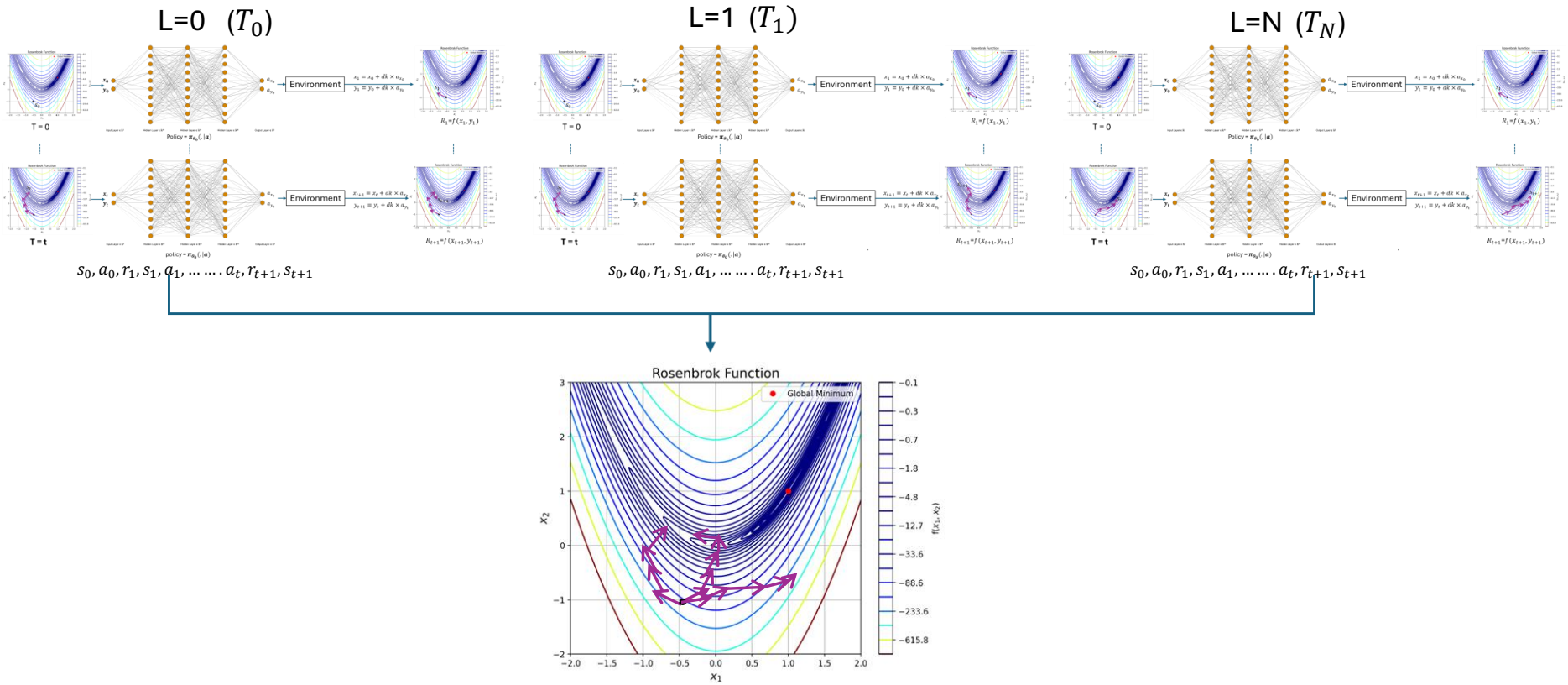
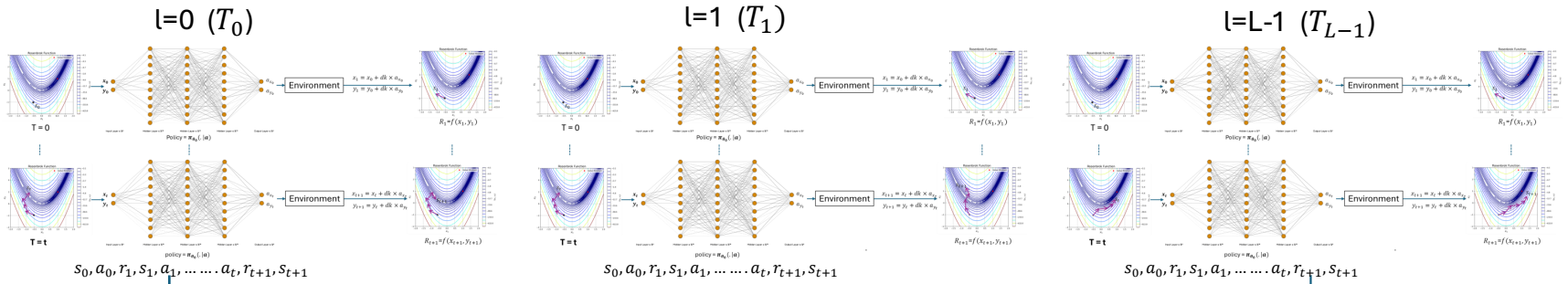
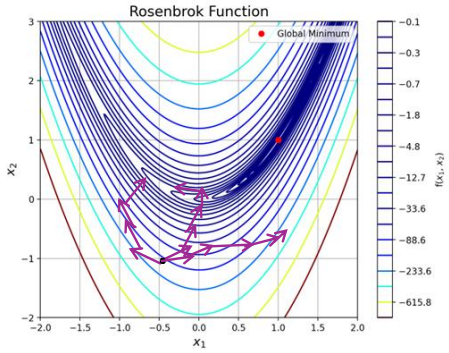


Figure: A single generation of the algorithm

Methodology

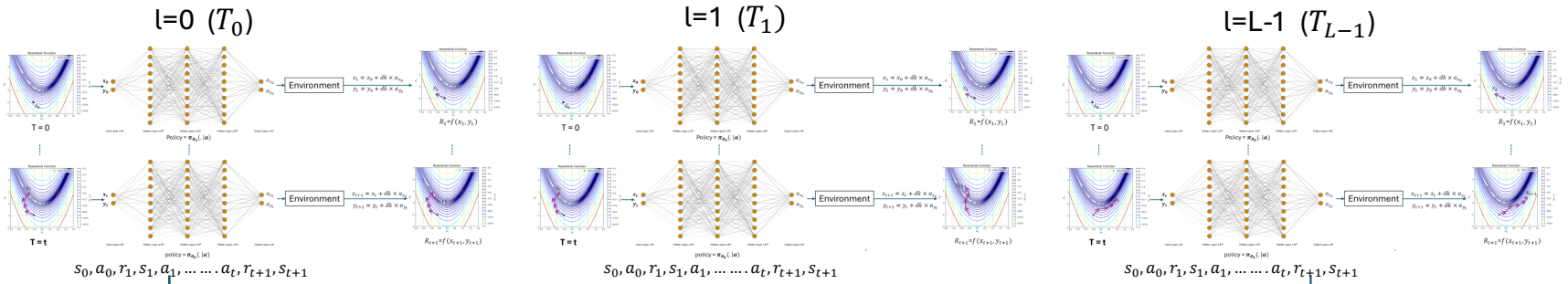


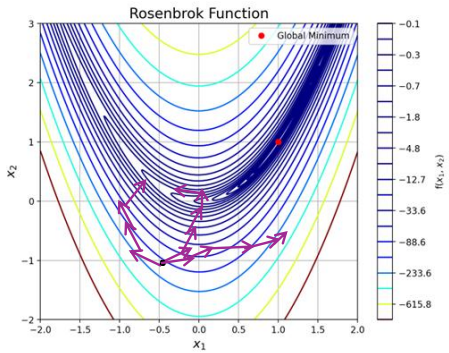


Update gradient of $\boldsymbol{\pi_{\theta_0}}(.|\boldsymbol{a})$ weights:

- **L** batches of states, actions and rewards trajectories $\left[\left\{\boldsymbol{s_t^l}, \boldsymbol{a_t^l}, \boldsymbol{r_t^l}\right\}_{\boldsymbol{t=0}}^{\boldsymbol{T}}\right]_{l=0}^{L}$

Methodology

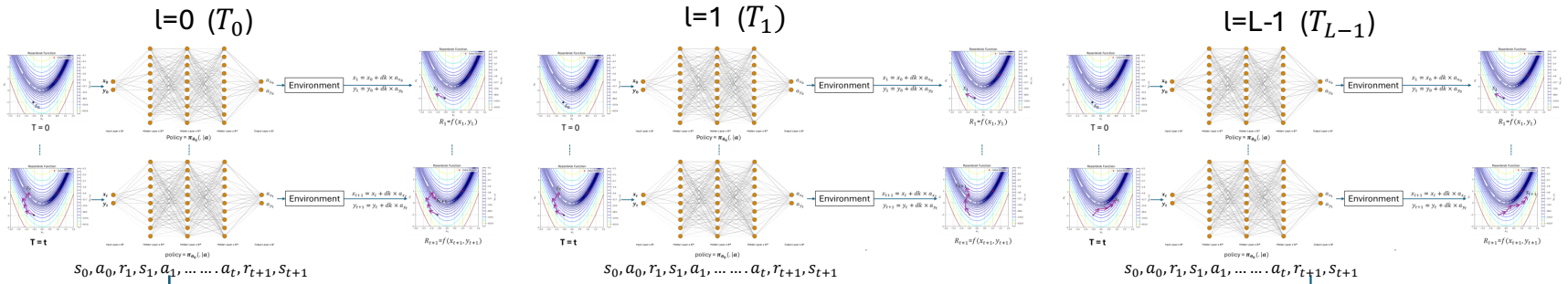


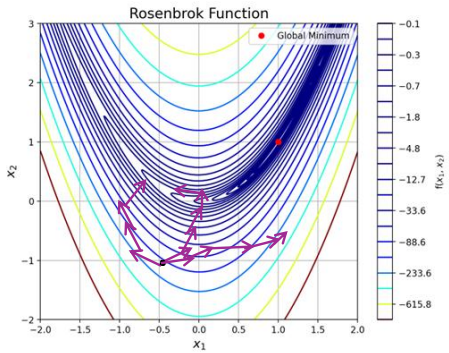


Update gradient of $\boldsymbol{\pi_{\theta_0}}(.|\boldsymbol{a})$ weights:

- **L** batches of states, actions and rewards trajectories $\left[\left\{\boldsymbol{s_t^l}, \boldsymbol{a_t^l}, \boldsymbol{r_t^l}\right\}_{\boldsymbol{t=0}}^{\boldsymbol{T}}\right]_{l=0}^{L}$
- Compute $R_T^l = \left\{\frac{1}{T}\sum_{t=0}^{T} r_t^l\right\}_{l=0}^{L}$

Methodology

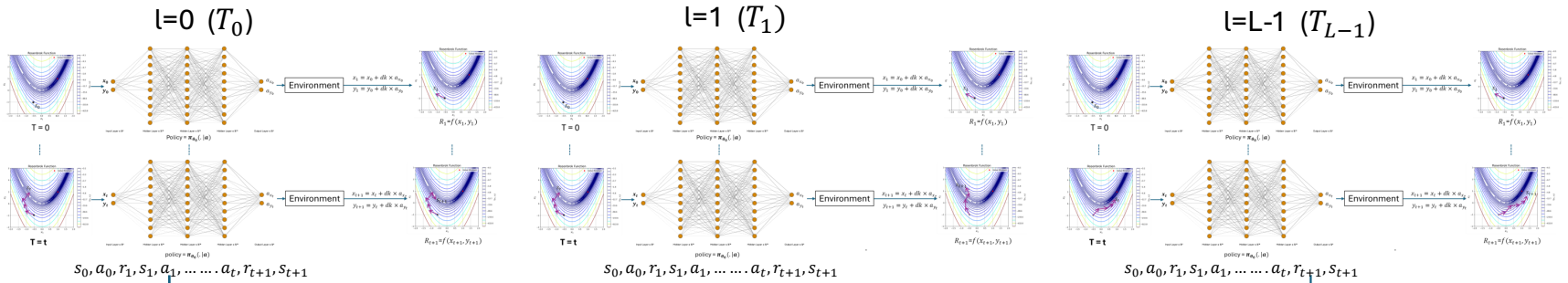


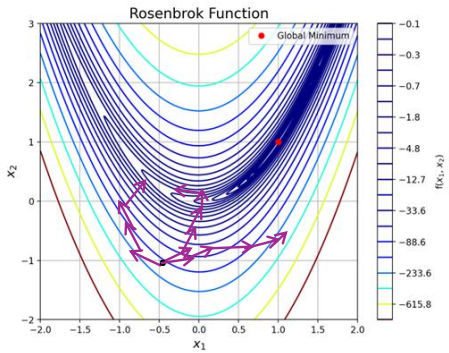


Update gradient of $\boldsymbol{\pi_{\theta_0}}(.|\boldsymbol{a})$ weights:

- **L** batches of states, actions and rewards trajectories $\left[\left\{\boldsymbol{s_t^l}, \boldsymbol{a_t^l}, \boldsymbol{r_t^l}\right\}_{\boldsymbol{t=0}}^{\boldsymbol{T}}\right]_{l=0}^{L}$
- Compute $R_T^l = \left\{\frac{1}{T}\sum_{t=0}^{T} r_t^l\right\}_{l=0}^{L}$
- $\hat{\nabla} J_T(\theta_0) = \frac{1}{L}\sum_{l=0}^{L-1}\left[R_T^l \sum_{t=0}^{T} \nabla \ln(\pi_{\theta_0}(\boldsymbol{a_t^l}|\boldsymbol{s_t^l})\right]$

Methodology





Update gradient of $\boldsymbol{\pi_{\theta_0}}(.|\boldsymbol{a})$ weights:

- **L** batches of states, actions and rewards trajectories $\left[\left\{\boldsymbol{s_t^l}, \boldsymbol{a_t^l}, \boldsymbol{r_t^l}\right\}_{\boldsymbol{t=0}}^{\boldsymbol{T}}\right]_{l=0}^{L}$
- Compute $R_T^l = \left\{\frac{1}{T}\sum_{t=0}^{T} r_t^l\right\}_{l=0}^{L}$
- $\widehat{\nabla} J_T(\theta) = \frac{1}{L}\sum_{l=0}^{L-1}\left[R_T^l \sum_{t=0}^{T} \nabla \ln(\pi_\theta(\boldsymbol{a_t^l}|\boldsymbol{s_t^l})\right]$

  N^{th} generation
- $\theta_{n+1} \coloneqq \theta_n + \alpha_n\left(\widehat{\nabla} J_T(\theta_n) - 2\beta\theta_n\right)$

Methodology

Parameters
Generations N
Initial state x_0
Step size dk
Trajectory length T
Batches L
Reward f(x)

Experiments and Results

Using RL-OPT:

- RL-OPT is implemented with a clear, class-based architecture.
- Reproduced the 1D and 2D benchmark results from the original paper.
- Extended optimization experiments to additional non-convex, non-differentiable 1D and 2D functions.
- Tuned key hyperparameters (e.g., episode length, learning rate).
- Built a lightweight policy network from scratch—to illustrate how the agent learns.
- Tried a reward-based stopping criterion to terminate training.

Experiments

Objective function used in RL-OPT:

$$f(x_1, x_2) = -(8 + \cos(10x_1) + \cos(10x_2) + 5(x_1^2 + x_2^2))$$

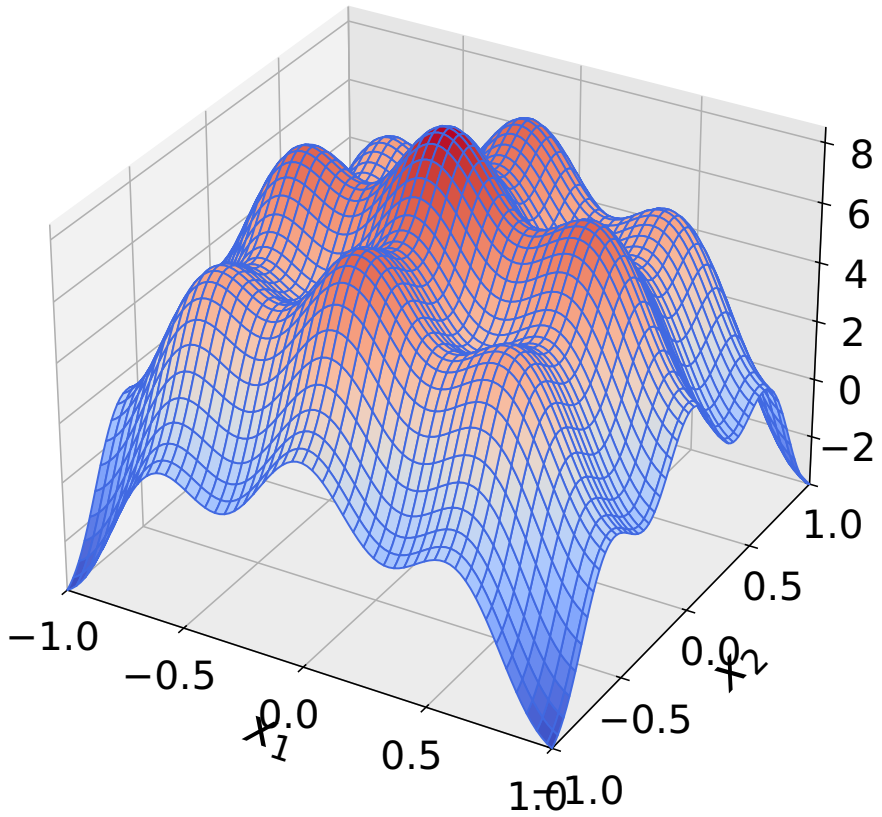
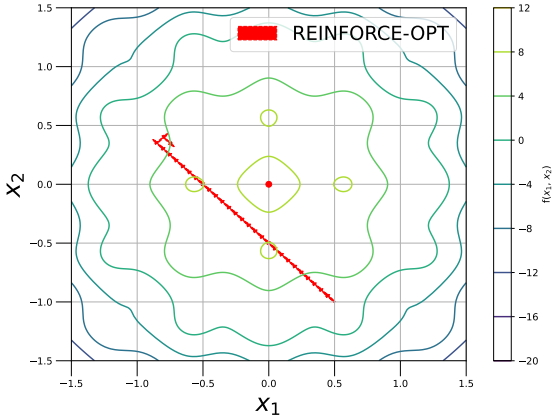


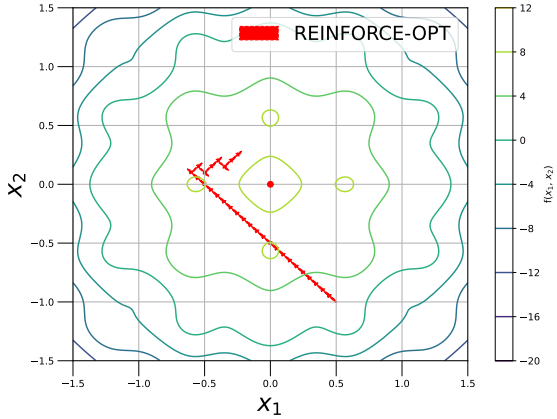
Table: REINFORCE Configuration

Parameter	**Value**
Generations	16 000
Initial state x_0	(0.5, –1)
Step size dk	0.05
Trajectory length T	30
Batches L	$10 \times 6 = 60$

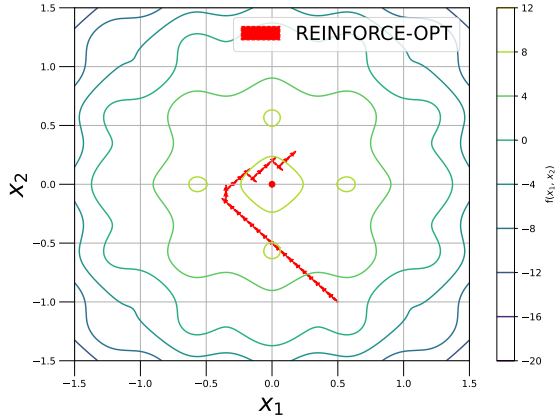
Experiments



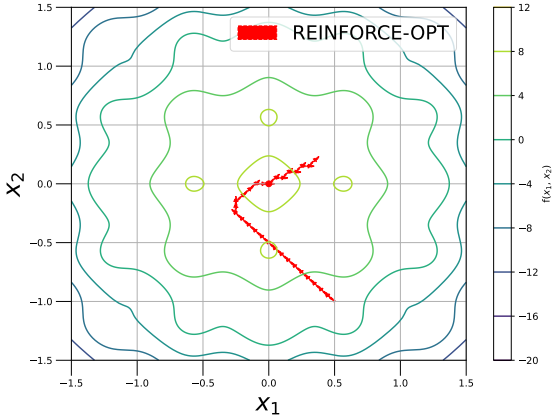
Plot 300



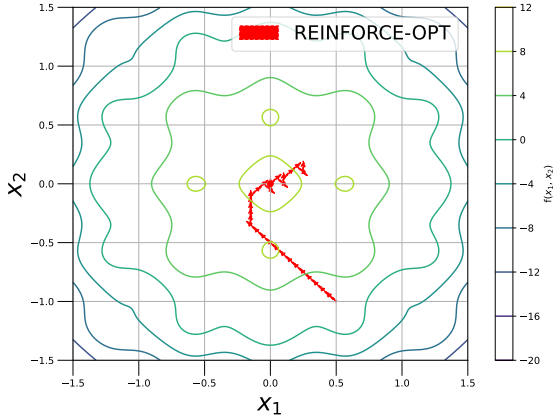
Plot 1600



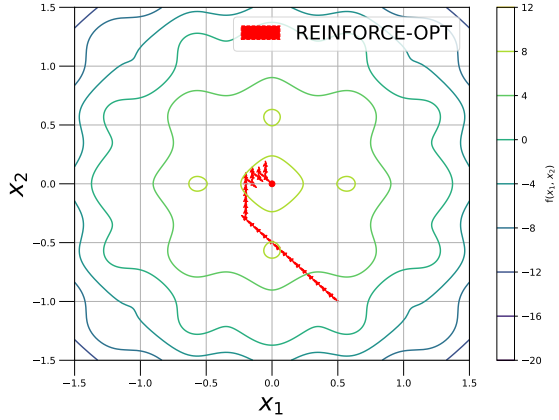
Plot 5000



Plot 7100



Plot 14300



Plot 15400

Experiments

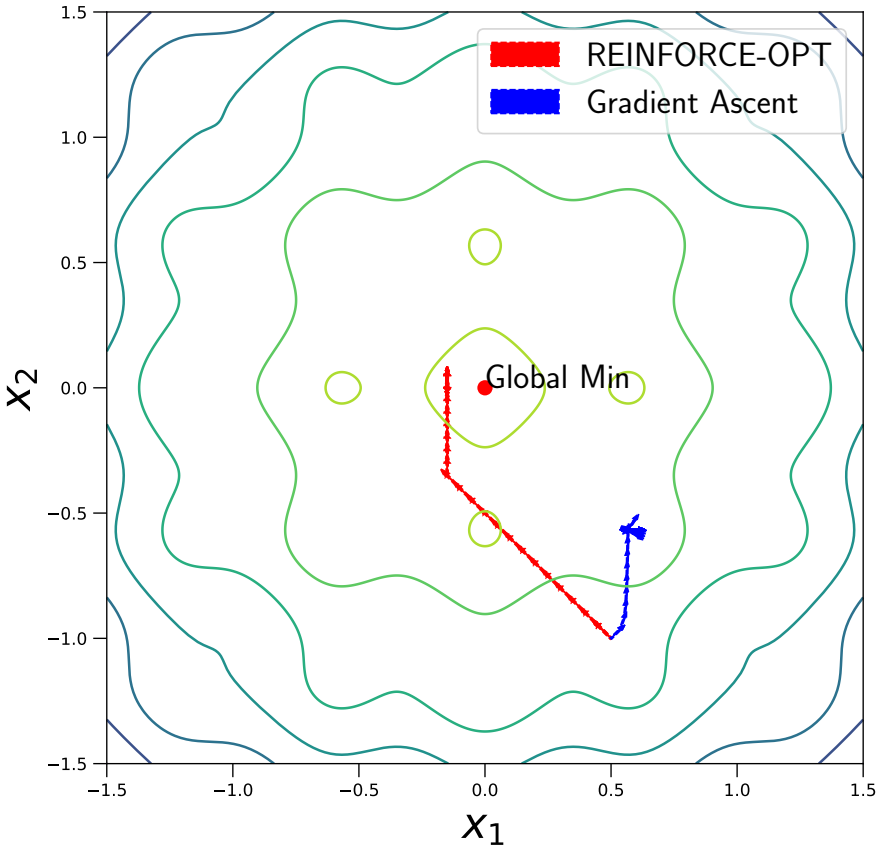


Figure: Gradient descent v/s Reinforce

Metric	Value
Best step	(0,0)
Best reward	10

Table: REINFORCE Optimization Results

Experiments

Optimization for Rosenbrock function:

$$f(x_1, x_2) = -((1 - x_1)^2 + 100(x_2 - x_1^2)^2)$$

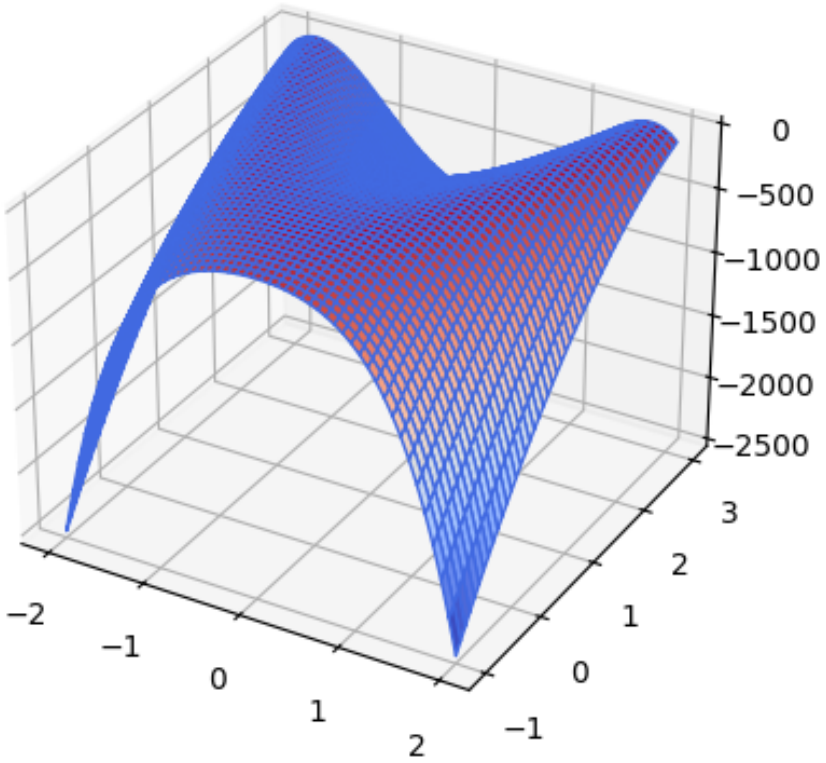
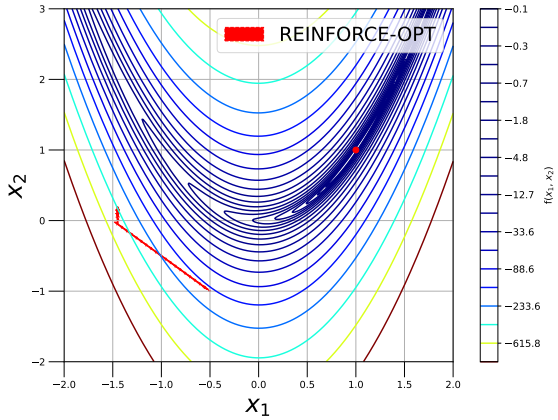


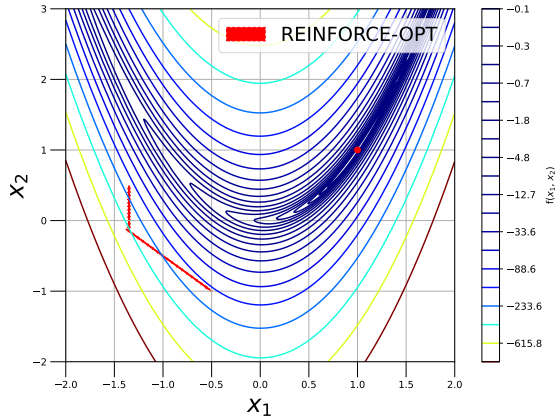
Table: REINFORCE Configuration

Parameter	**Value**
Generations	16 000
Initial state x_0	(-0.5, –1)
Step size dk	0.05
Trajectory length T	30
Batches L	$10 \times 6 = 60$

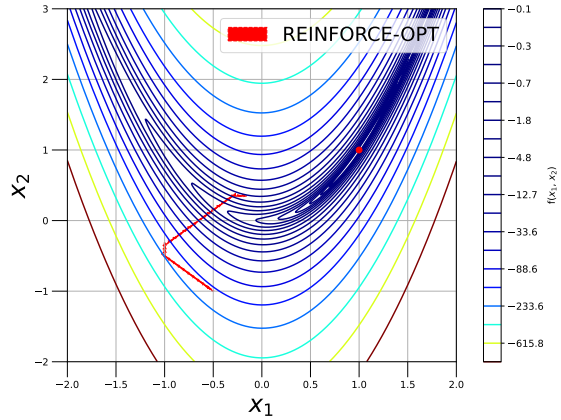
Experiments: Rosenbrock Function



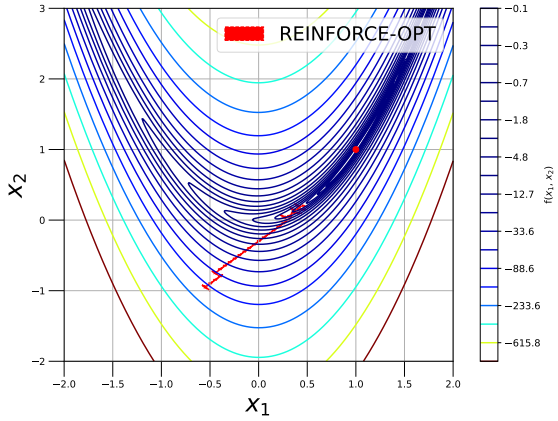
Plot 100



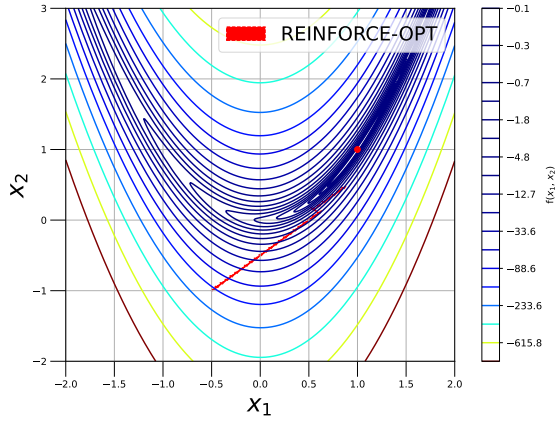
Plot 1000



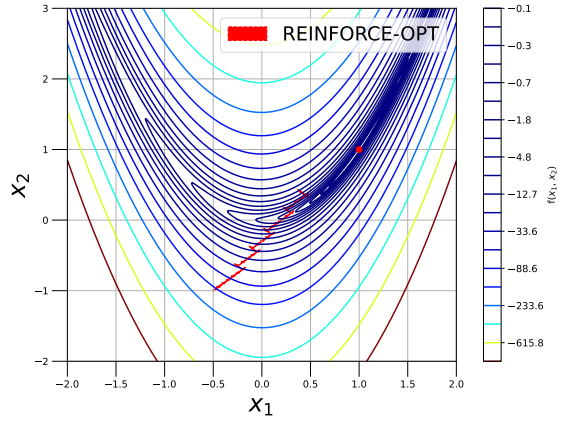
Plot 3400



Plot 5600



Plot 10800



Plot 16000

Experiments

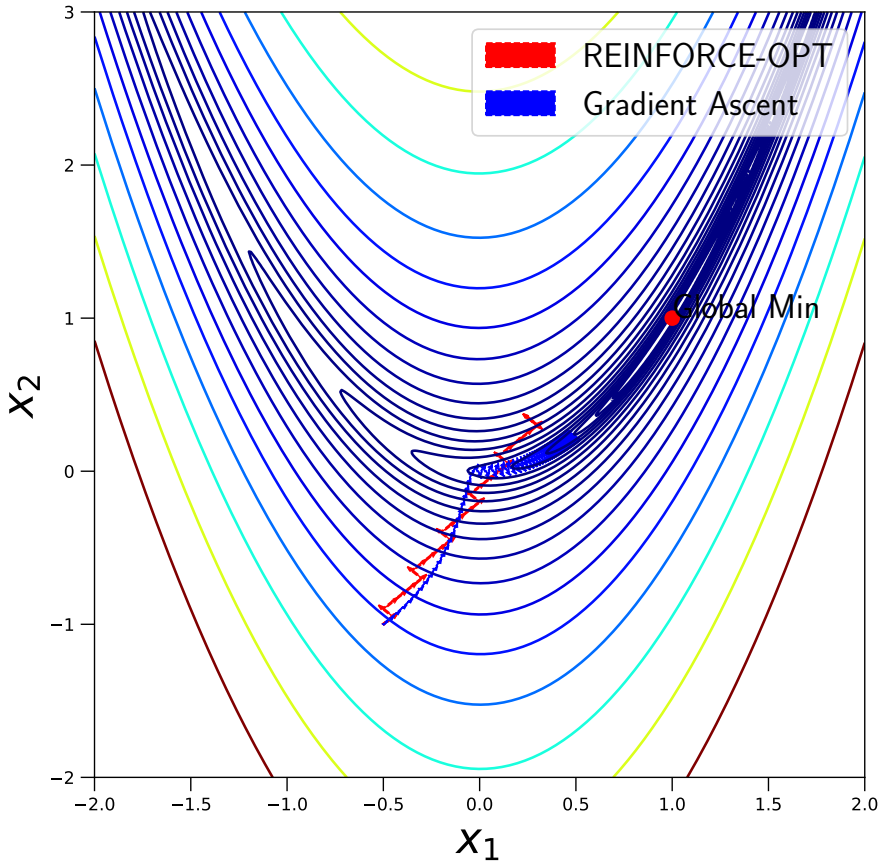


Figure: Gradient descent v/s Reinforce

Metric	Value
Best step	(1,1)
Best reward	0

Table: REINFORCE Optimization Results

Experiments

Optimization for Absolute value function:

$$f(x_1, x_2) = |x_1 - 1| + 2\,|x_2 - 2|$$

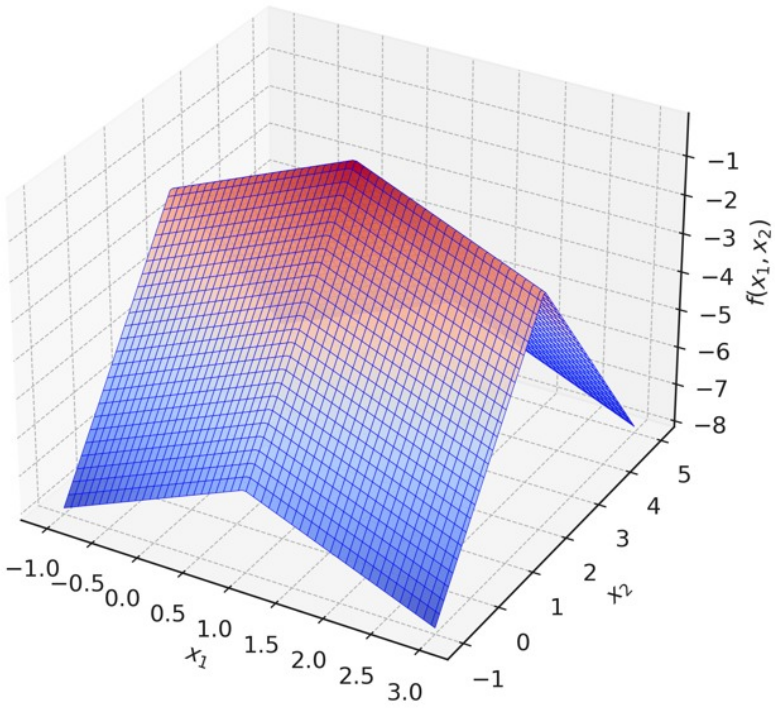
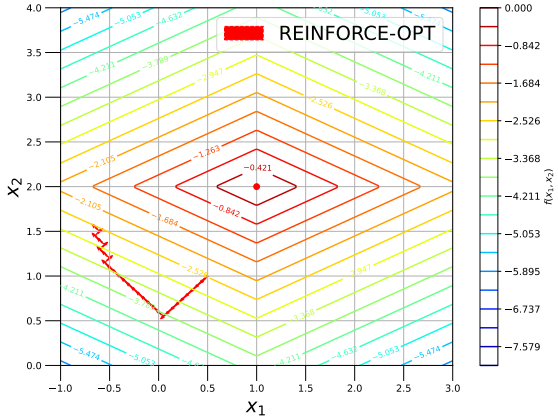


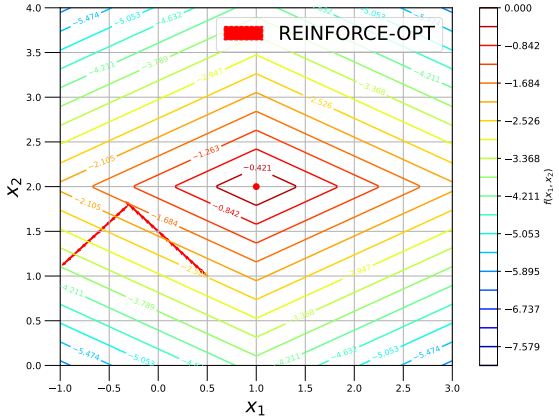
Table: REINFORCE Configuration

Parameter	**Value**
Generations	7 100
Initial state x_0	(0.5, 1)
Step size dk	0.05
Trajectory length T	30
Batches L	$10 \times 6 = 60$
Global maximum: $(1, 2)$	
Best reward: 0	

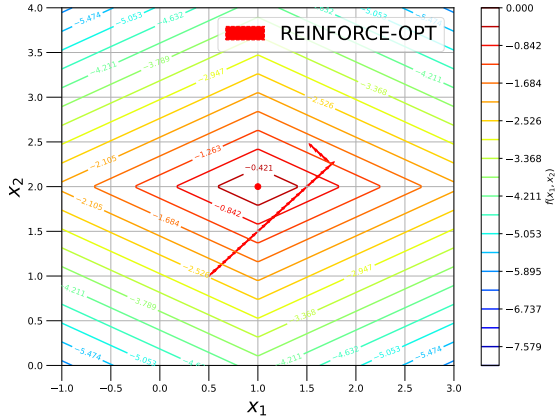
Experiments: Absolute value function



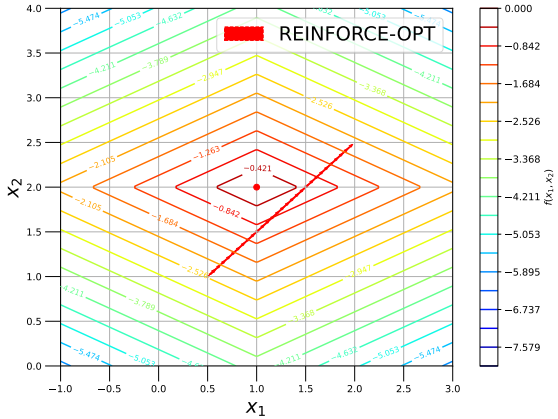
Plot 100



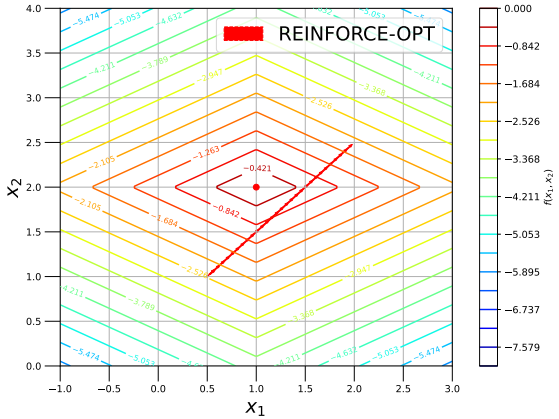
Plot 700



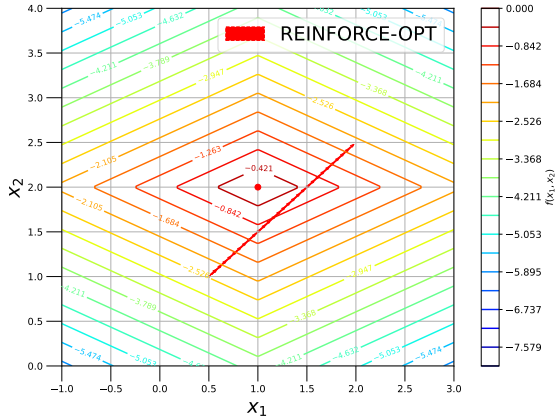
Plot 1100



Plot 1600



Plot 5200



Plot 7100

Conclusion (1/2) – Achievements and Significance

- **Objective:** Applied **REINFORCE-OPT**, a policy-gradient RL method, to optimize **low-dimensional functions** 1D and 2D without gradient information.
- **Main Results:**
  - After training for several thousand generations, the agent reliably produced high-quality solutions.
  - Effective even on **highly non-convex functions** where traditional methods stagnate.
- **Computation Cost (on Mac Pro, no GPU):**
  - 1D function: 7 hours with 3500 generations
  - 2D Rosenbrock function: 10 hours with 60000 generations

  Parallel environments, episode length, and step size significantly affect performance.

Conclusion (2/2) – Stopping Criteria and Limitations

- **Problem:** Fixed generation stopping rule offers no guarantee of policy convergence.
- Training may continue **after performance has plateaued**, wasting computational resources.
- **Proposed Early-Stopping Criteria:**
  1. **Weight Norm Criterion:** Stop if ℓ_2 norm change of weights is below ε_w (e.g., 10^{-6}) for 2000 generations.
  2. **Reward Improvement Criterion:** Stop if best reward doesn't improve by at least ε_r (e.g., 10^{-3}) for 2000 generations.
- **Risk:** If the global optimum is far from recent solutions, policy may **prematurely converge**.

Future Work

1. **Higher-dimensional extension:**
   - Apply RL-OPT to more challenging high-dimensional functions.
   - Analyze the effects of generations, step size, and batch size with more functions.
   - Experiment with different network configurations (e.g., depth, activation functions).
2. **Early-stopping implementation:**
   - Fully integrate and evaluate the proposed criteria.
3. **Fix it issue by issue:**
   - Need to explore it for the sparse deconstruction problem.
4. **Environment upgrade:**
   - Switch from TensorFlow to **PyTorch** for better GPU support in parallel environments.
5. **Continuous action spaces:**
   - Extend from log-probability-based discrete actions to **Gaussian-modeled continuous actions**.

Thank you for your attention!

Questions are welcome.

1. Barto Andrew and Sutton Richard S. Reinforcement learning: an introduction. 2018.
2. Chen Xu, Yun-Bin Zhao, Zhipeng Lu, and Ye Zhang. Reinforcement-learning- based algorithms for optimization problems and applications to inverse problems