-4391C

Reinforcement Learning for PDE Control Problems

Nick Winovich[†], Bart van Bloemen Waanders[†]
Deepanshu Verma[‡], Lars Ruthotto[‡]

[†] Sandia National Laboratories[1]
Center for Computing Research
Scientific Machine Learning Group

[‡] Emory University
Department of Mathematics
Department of Computer Science
Scientific Computing Group

SIAM-UQ 2022

---

[1] Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned

dia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly ow
subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Overview

1 PDE Control Problems

- Motivating Examples
- Control Theory
- Prototype PDE System

2 Reinforcement Learning

- Actor-Critic Agent Models
- Proximal Policy Optimization
- Numerical Results

Overview

1 PDE Control Problems

- Motivating Examples
- Control Theory
- Prototype PDE System

2 Reinforcement Learning

- Actor-Critic Agent Models
- Proximal Policy Optimization
- Numerical Results

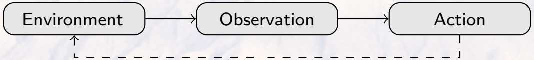
Control Problems in Scientific Computing

Control problems encapsulate a wide range of applications in scientific computing relevant to large-scale issues such as:

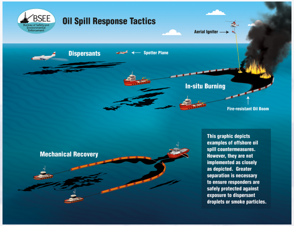
- Containment of wildfires, response to natural disasters
- Reducing impact of oil spills and contamination events

High-level points:

- Need for policy guidance/assistance with real-time decisions
- Control problems often arise naturally in the form of a sequence of “action-reaction” iterations:



Environment → Observation → Action



This graphic depicts examples of offshore oil spill countermeasures. However, they are not implemented as closely as depicted. Greater separation is necessary to ensure responders are safely protected against exposure to dispersant droplets or smoke particles.

PDE Control Problem Setup

Find an *optimal control policy* $a^*(t) = P(u(t), t)$ that induces a trajectory u^* of the system $\frac{\partial u}{\partial t}(t) = G(u(t), a(t), t)$ which minimizes the cost function:

$$J(t) = I(u(T), T) + \int_t^T L(u(t), a(t), t)\, dt \tag{1}$$

e.g. with a spatial loss defined by $L = \int_{\Omega_T} |u^+|^2\, dm$ for a given target region $\Omega_T \subset \Omega$.

Given parameters θ, find a policy $a(x, y, t)$ such that the solution $u(x, y, t)$ to:

$$\begin{cases} \mathcal{L}_\theta[u] = F_\theta[a] & \text{in} \quad \Omega \times [0, T] \\ u = g_d & \text{on} \quad \Omega_d \\ \frac{\partial u}{\partial n} = g_n & \text{on} \quad \Omega_n \end{cases} \tag{2}$$

minimizes the cost function $J = \int_0^T \int_{\Omega_T} |u^+|^2\, dm\, dt$ for a given target region $\Omega_T \subset \Omega$.

Control Problems in Scientific Computing

- **Computational Time:** fast evaluation of policy required for real-time applications.
- **Scale:** states are governed by PDEs → *infinite-dimensional*
- **Uncertainty:** precise source location and velocity field are unknown.
- **Local vs. Global solution method:**
    - **Local Method:** optimal policy for *fixed* problem configuration, e.g. NLP methods, PMP.
    - **Global Method:** optimal policy for *every* problem configuration, e.g. Dynamic Programming (HJB PDE), *Reinforcemt Learning (RL)* ; suitable for real-time applications
- **Curse-of-Dimensionality:** HJB solvers suffer from CoD→ cost increases with dimension

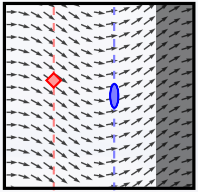
Prototype PDE Environment

PDE System

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u - D \cdot \Delta u = f^{(t)} \quad \text{in} \quad \Omega \times [0, T] \quad \text{with} \quad D = 0.5$$

$$u = 0 \;\text{ on }\; \{x = 0\} \cup \{y = 0\} \cup \{y = 1\} \quad \text{and} \quad \frac{\partial u}{\partial n} = 0 \;\text{ on }\; \{x = 1\}$$

Environment



Initial Source

$$f^{(0)} = 5.0/\sigma \cdot \exp\left(-(|x - x_0| + |y - y_0|)/\sigma\right) \quad \text{with} \quad \sigma = 0.01$$

$$x_0 \sim \text{Uniform}(0.1, 0.25) \quad \text{and} \quad y_0 \sim \text{Uniform}(0.1, 0.9)$$

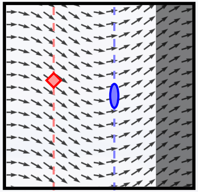
Velocity Field

$$\mathbf{v}_\delta(x, y) = \left(\sqrt{\eta^2 - \delta^2 \cdot \sin^2(2\pi \cdot [x - \phi_0])}\;,\; -\eta \cdot \sin(2\pi \cdot [x - \phi_0]) \right)$$

where $\eta = 12.5$, $\delta = 0.75$, and $\phi_0 \sim \text{Uniform}(0.0, \text{-}0.3)$

Prototype PDE Environment

Environment



Sink Update

$$f^{(t+1)} = f^{(t)}(x,y) - \alpha \cdot \exp\left(-(|x-0.5|/\sigma_x + |y-A_t|/\sigma_y)\right)$$

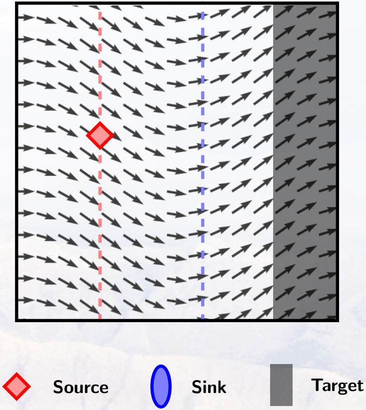
with $\sigma_x = 0.025$, $\sigma_y = 0.05$, and $\alpha = 2.5$

- Update RHS every $\Delta t = 0.02$ time-step (25 total updates)

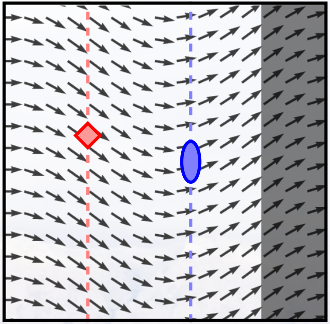
Control Decision

"Select the y-coordinate $A_t \in [0,1]$ of the next sink location based on the current system state $U_t = \{u(x,y,t)\}_{(x,y)\in\Omega}$"

Prototype PDE Environment: Time-step 0

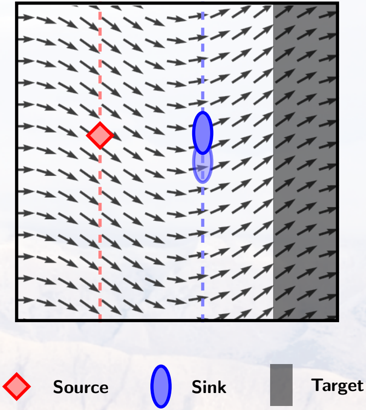


Prototype PDE Environment: Time-step 1

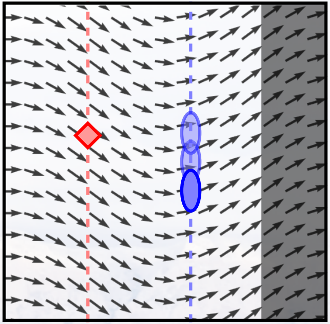


◆ **Source** ⬮ **Sink** ▮ **Target**

Prototype PDE Environment: Time-step 2

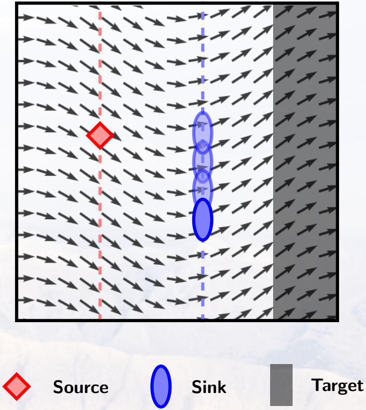


Prototype PDE Environment: Time-step 3



◆ **Source** ⬮ **Sink** ■ **Target**

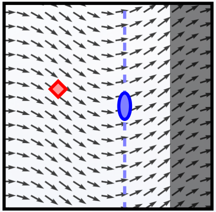
Prototype PDE Environment: Time-step 4



PDE Control Problem: Local vs. Global Solutions

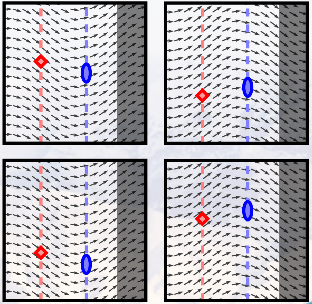
Local Solutions

- Fixed source location and velocity field
- Solutions are computed w.r.t. a fixed problem instance



Global Solutions

- Varying source location and velocity field
- Offline calibration yields approximate solutions for an entire family of problem instances
- Minimal overhead for performing run-time inference



Overview

1. PDE Control Problems
    - Motivating Examples
    - Control Theory
    - Prototype PDE System

2. Reinforcement Learning
    - Actor-Critic Agent Models
    - Proximal Policy Optimization
    - Numerical Results

Reinforcement Learning

Environment

The dynamics of a PDE control problem can be used to define the transition map and reward function for a MDP.

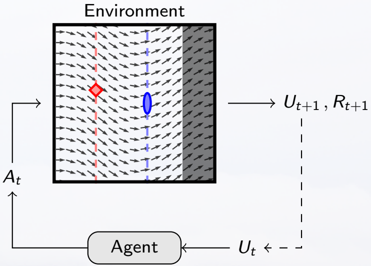
Observations

Decisions are often made based off of incomplete information regarding the underlying physical system.

Agent

The agent processes observation data in order to select an appropriate action for achieving the best possible outcome.

- The agent's policy is parameterized by values θ corresponding to a neural network architecture.
- This leads to a differentiable policy assignment rule.



- Observation data U_t may differ from the internal system state S_t
- Details regarding the calculation of the reward R_{t+1} are unknown to the agent

Reinforcement Learning: Objective

Objective Function

Given a parameterized policy π_θ and initial state distribution d_0, we define the *episodic-return objective*:

$$J(\theta) \;=\; \mathbb{E}_{\pi_\theta}\left[\,\sum_{t=0}^{T} \Delta t \cdot R_{t+1} \,\middle|\, S_0 \sim d_0\right]$$

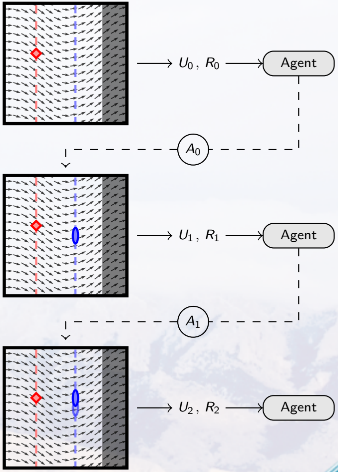
Policy Gradient Theorem

The gradient of the objective function can be expressed as follows:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\pi_\theta}\left[\,\sum_{t=0}^{T} \Delta t \cdot q_{\pi_\theta}(S_t, A_t) \cdot \nabla_\theta \log \pi_\theta(A_t|U_t) \,\middle|\, S_0 \sim d_0\right]$$

where $q_\pi(s,a) \;=\; \mathbb{E}_{\pi_\theta}[G_t|S_t = s, A_t = a]$.

- Optimal parameters are then approximated using gradient ascent.



Reference: https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021 [Lecture 9]

Reinforcement Learning: Actor-Critic Agents

Variance Reduction

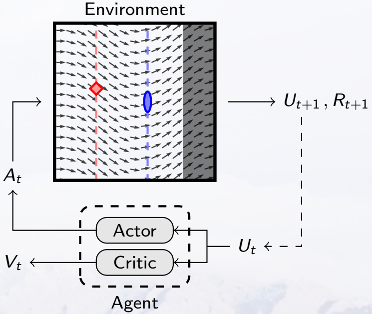
$$\mathbb{E}_{\pi_\theta}\left[\, B(U_t) \cdot \nabla_\theta \log \pi_\theta(A_t|U_t) \,\right] \;=\; 0$$

for any baseline $B(U_t)$ independent of the choice of action A_t.

For a value estimate V_{π_θ} dependent only on the current state, we have:

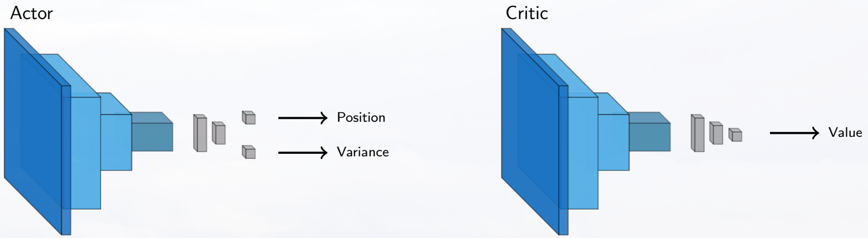
$$\begin{aligned}\nabla_\theta J(\theta) \;&=\; \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \Delta t \cdot \left[q_{\pi_\theta}(S_t, A_t) - V_{\pi_\theta}(U_t) \right] \cdot \nabla_\theta \log \pi_\theta(A_t|U_t) \right] \\ &\approx\; \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \Delta t \cdot \left[G_t - V_{\pi_\theta}(U_t) \right] \cdot \nabla_\theta \log \pi_\theta(A_t|U_t) \right]\end{aligned}$$

- Variance in the gradient approximation can be significantly reduced by providing an accurate value estimate.
- The action-value can be approximated by the observed return G_t.



Reference: https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021 [Lecture 9]

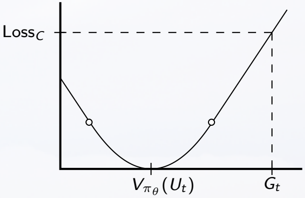
Actor-Critic Neural Networks



- Both the actor and critic network architectures begin with convolutional layers to efficiently parse the spatially-structured observation data retrieved from the PDE environment.
- The actor provides predictions for the parameters of a probability distribution characterizing the desired action, while the critic network estimates the value of the current state.

Critic Loss Definition

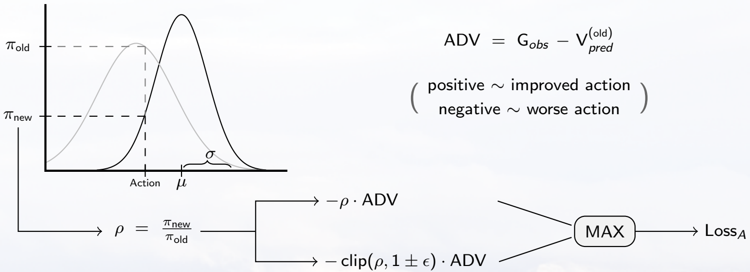
$$\nabla_\theta J(\theta) \;\approx\; \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \Delta t \cdot \Big[\underset{\substack{\uparrow \\ \text{observed return}}}{G_t} - \underset{\substack{\uparrow \\ \text{critic prediction}}}{V_{\pi_\theta}}(U_t)\Big] \cdot \nabla_\theta \log \pi_\theta(A_t|U_t)\right]$$



$$\text{Loss}_C \;=\; \Big|\underbrace{G_t - V_{\pi_\theta}(U_t)}_{\text{“advantage”}}\Big|^2$$

- The critic is trained to provide an accurate baseline for variance reduction.
- Huber Loss can be used to improve robustness of estimate with respect to outliers.

Actor Loss: Proximal Policy Optimization



$$\rho = \frac{\pi_{\text{new}}}{\pi_{\text{old}}}$$

$$-\rho \cdot \text{ADV}$$

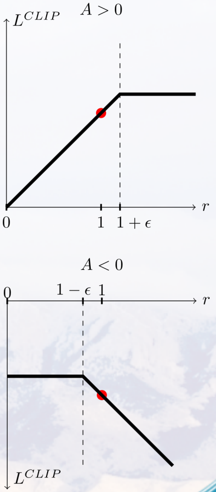
$$-\,\text{clip}(\rho, 1 \pm \epsilon) \cdot \text{ADV}$$

MAX

Loss_A

$$\text{ADV} = \text{G}_{obs} - \text{V}^{(\text{old})}_{pred}$$

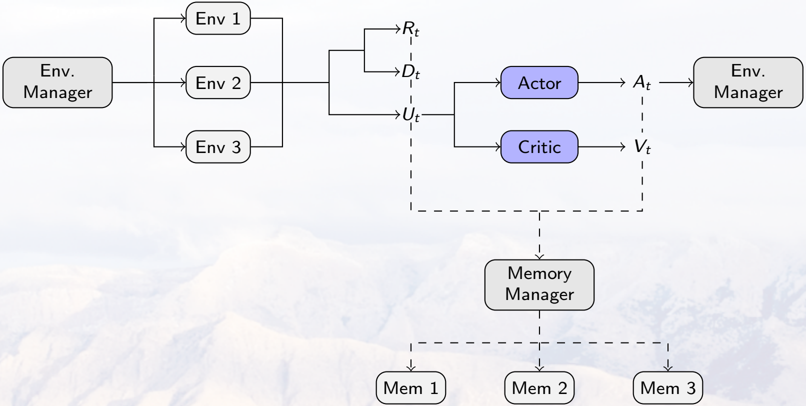
$$\left(\begin{array}{c}\text{positive} \sim \text{improved action}\\ \text{negative} \sim \text{worse action}\end{array}\right)$$



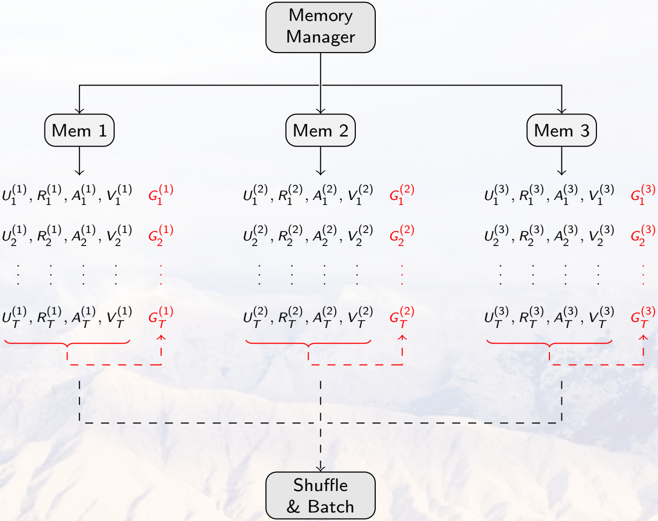
- PPO is designed to avoid over-tuning parameters so that the updated actor distributions remain relatively close to the previous distributions.
- Simplified version of previous work on *trust region policy optimization*.

Reference: Proximal Policy Optimization Algorithms [https://arxiv.org/pdf/1707.06347.pdf]

Parallel Environment Workflow



Parallel Environment Workflow



Memory Manager

Mem 1

$U_1^{(1)}, R_1^{(1)}, A_1^{(1)}, V_1^{(1)} \quad G_1^{(1)}$

$U_2^{(1)}, R_2^{(1)}, A_2^{(1)}, V_2^{(1)} \quad G_2^{(1)}$

$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$

$U_T^{(1)}, R_T^{(1)}, A_T^{(1)}, V_T^{(1)} \quad G_T^{(1)}$

Mem 2

$U_1^{(2)}, R_1^{(2)}, A_1^{(2)}, V_1^{(2)} \quad G_1^{(2)}$

$U_2^{(2)}, R_2^{(2)}, A_2^{(2)}, V_2^{(2)} \quad G_2^{(2)}$

$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$

$U_T^{(2)}, R_T^{(2)}, A_T^{(2)}, V_T^{(2)} \quad G_T^{(2)}$

Mem 3

$U_1^{(3)}, R_1^{(3)}, A_1^{(3)}, V_1^{(3)} \quad G_1^{(3)}$

$U_2^{(3)}, R_2^{(3)}, A_2^{(3)}, V_2^{(3)} \quad G_2^{(3)}$

$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$

$U_T^{(3)}, R_T^{(3)}, A_T^{(3)}, V_T^{(3)} \quad G_T^{(3)}$

Shuffle & Batch

Parallel Environment Algorithm

Algorithm 1 Actor-Critic Agent: Proximal Policy Optimization

```
for $n$ in $\{1, \ldots, \texttt{num\_episodes}\}$ do
    environment_manager.reset()                                          ▷ Run simulations
    states, actions, values, rewards ← run_parallel_envs()

    returns ← memory_manager.compute_returns(rewards)                    ▷ Process results
    memory_manager.shuffle_and_batch()

    for $k$ in $\{1, \ldots, \texttt{num\_batches}\}$ do
        ADV ← (returns[k] − values[k])                                   ▷ Evaluate actions
        ratio ← $\pi_{\theta^A}$(actions[k] | states[k]) / probs[k]
        clipped ← $-$ADV · clip(ratio, $1 - \epsilon$, $1 + \epsilon$)
        unclipped ← $-$ADV · ratio

        $L_A \leftarrow$ max[clipped, unclipped]                             ▷ Compute losses
        $L_C \leftarrow \frac{1}{2}\texttt{ADV}^2$   or   huber_loss(values[k], returns[k])
        minimize($L_A, \theta^A$)                                            ▷ Update parameters
        minimize($L_C, \theta^C$)
    end for
end for
```

Initial Predictions for Control Policies

Rewards

-70.70	-36.21	-20.66
-87.85	-47.75	-28.22
-66.77	-54.20	-53.73

Network Predictions for Control Policies

Rewards

-66.09	-32.51	-14.10
-61.22	-36.60	-25.22
-31.11	-38.41	-47.21

Concluding Remarks

Summary

- RL provides a potential framework for solving PDE control problems semi-globally.
- Run-time inference requires minimal overhead after offline calibration.
- Parallel implementations can be leveraged to reduce training time.

Future Work

- How well does this approach extend to more complex systems?
- How does this compare with local methods and semi-global dynamic programming?
- Can knowledge of the physical system be incorporated to improve model calibration?
    - HJB equations, FEM formulation

Questions

Thank you for your time.

Questions?