Comparing Reinforcement Learning to Optimal Control Methods on the Continuous Mountain Car Problem

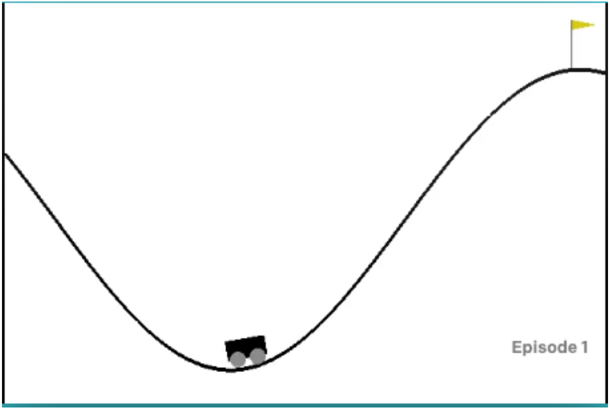
D Chowdhury, J Mantooth, A Sethi-Olowin

Mentor: Lars Ruthotto

2022 NSF REU/RET Computational Mathematics for Data Science

July 19, 2022

Continuous mountain car scenario



Continuous mountain car as optimal control problem

- The continuous mountain car can be modeled as

$$\partial_t \mathbf{z}(s) = \begin{pmatrix} z_2 \\ -\mu \cos(3z_1) \end{pmatrix} + \begin{pmatrix} 0 \\ \gamma \end{pmatrix} u(s), \quad \mathbf{z}(t) = z_t, \tag{1}$$

for $s \in (t, T]$ with the constants $\mu = 0.0025$, $\gamma = 0.0015$ and $\mathbf{z}(s) = \left(z_1, z_2\right)$.

- The unknown control $u : [t, T] \to [-1, 1]$ specifies the acceleration

- We seek to minimize the objective functional

$$J_{s,\mathbf{z}}[u] = \int_t^T L(s, \mathbf{z}, u)\, ds + g(\mathbf{z}(T)) \tag{2}$$

The running cost: $L(s, \mathbf{z}, u) = 0.01\|u(s)\|^2$
The terminal cost:

$$g(z) = \begin{cases} 0.45 - z_1, & z_1 < 0.45 \\ 0, & \text{otherwise.} \end{cases}$$

Why choose the mountain car?

- Established benchmark for RL models
- 2-D state-space allows for good plots and visualizations
- Both RL and optimal control problem
- Finite horizon (time)
- Continuous state and motion

Three Approaches:

1. Local solution using numerical ODE solvers and nonlinear optimization (baseline)
2. Reinforcement learning with actor-critic algorithm (data-based approach)
3. Optimal control using both model and data

Local Solution using Numerical ODE Solvers and Nonlinear Optimization

- To find the optimal control $\mathbf{u}_h$ we formulate an optimization problem
- We first discretize the control, state and the Lagrangian
- Setting $z_h^{(0)} = z_t$ and $\ell_h^{(0)} = 0$ allows us to use a forward Euler scheme for some control u

$$\begin{pmatrix} \mathbf{z}_h^{(i+1)} \\ \ell_h^{(i+1)} \end{pmatrix} = \begin{pmatrix} \mathbf{z}_h^{(i)} \\ \ell_h^{(i)} \end{pmatrix} + h \begin{pmatrix} f(t_i, \mathbf{z}_h^{(i)}) + \mathbf{B}\mathbf{u}_h^{(i)} \\ L(t_i, \mathbf{z}_h^{(i)}, \mathbf{u}_h^{(i)}) \end{pmatrix}, \quad i = 0, \ldots, N-1.$$

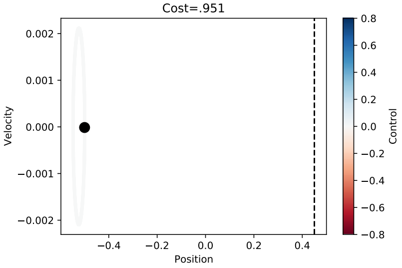
- Approximate our objective function as

$$J_{\mathbf{z}}[u] \approx J_{\mathbf{z}_h}(\mathbf{u}_h) = \ell_N + g(\mathbf{z}_h^{(N)})$$

- Yields the optimization problem

$$\min_{\mathbf{u}_h, \mathbf{z}_h} \ell_N + g(\mathbf{z}_h^{(N)})$$

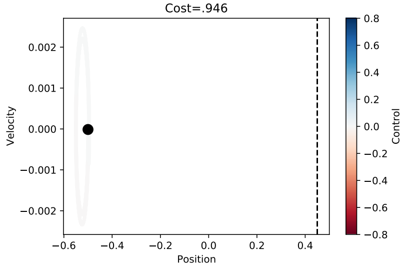
Gradient descent and corresponding trajectory plot



- To solve our optimization problem we use gradient descent. Take an initial guess for u_h and repeatedly update u_h using the gradient of the objective function and step size α

$$(u_h)_0 = \vec{0}$$

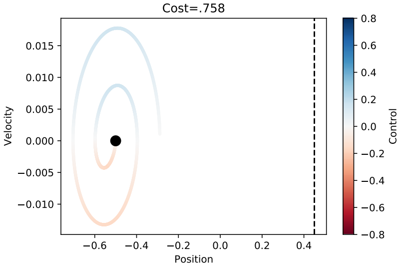
Gradient descent and corresponding trajectory plot



- To solve our optimization problem we use gradient descent. Take an initial guess for u_h and repeatedly update u_h using the gradient of the objective function and step size α

$$(u_h)_0 = \vec{0}$$
$$(u_h)_1 = (u_h)_0 - \alpha(\nabla J((u_h)_0))$$

Gradient descent and corresponding trajectory plot



- To solve our optimization problem we use gradient descent. Take an initial guess for u_h and repeatedly update u_h using the gradient of the objective function and step size α

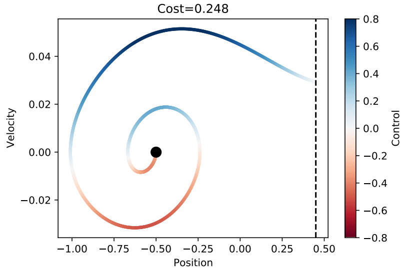
$$(u_h)_0 = \vec{0}$$

$$(u_h)_1 = (u_h)_0 - \alpha(\nabla J((u_h)_0))$$

$$\vdots$$

$$(u_h)_6 = (u_h)_5 - \alpha(\nabla J((u_h)_5))$$

Gradient descent and corresponding trajectory plot



- To solve our optimization problem we use gradient descent. Take an initial guess for u_h and repeatedly update u_h using the gradient of the objective function and step size α

$$(u_h)_0 = \vec{0}$$

$$(u_h)_1 = (u_h)_0 - \alpha(\nabla J((u_h)_0))$$

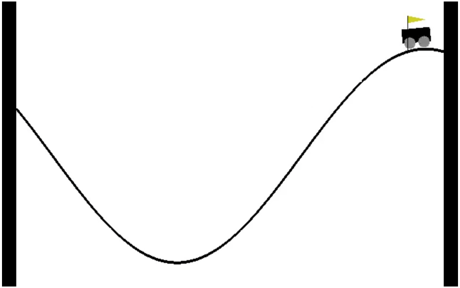
$$\vdots$$

$$(u_h)_6 = (u_h)_5 - \alpha(\nabla J((u_h)_5))$$

$$\vdots$$

$$(u_h)_* = (u_h)_{19} - \alpha(\nabla J((u_h)_{19}))$$

Optimal solution rendered (using local method)



Issues with the local method

- Dependent on starting location
- Unable to adapt to environment changes midway through
- Non-linear and non-convex problem which makes the method slower

Reinforcement learning

- Data-based approach to find global policy (for any initial condition)
- RL only considers the objective function and has no knowledge of the model
- Caveat: RL likes maximizing rewards so we will maximize negative cost
- Stochastic in two ways: initial position and action space
  - Allows for exploration
- We aim to estimate an optimal control policy $\psi : [t, T] \times \mathbb{R}^d \to \mathbb{R}^n$ via

$$\min_{\psi} \mathbb{E}_{z_t \sim \rho} \left(J_{\mathbf{z}}[u] \right) \quad \text{subject to} \quad (1).$$

TD-Advantage Actor-Critic

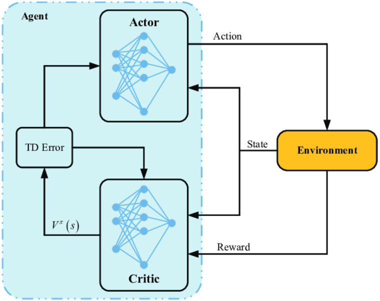


Figure: Actor-Critic model [1]

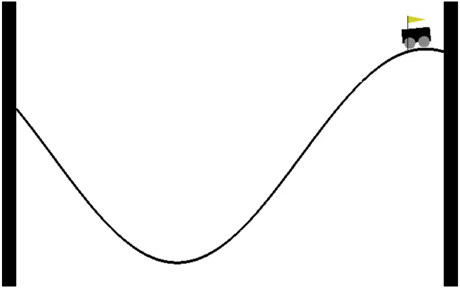
- Actor: $\psi^U(s, z_s) \approx \psi(s, z_s)$
- Critic: $V_\psi^V(z_s) \approx \mathbb{E}[J_{z_s}[\psi]]$
- TD Error:
  $\delta = r_{s+1} + V^\psi(z_{s+1}) - V^\psi(z_s)$
- Online algorithm
- Learns only from observations (data)

[1] H. Giang, T. Hoan, P. Thanh, and I. Koo. Hybrid noma/oma-based dynamic power allocation scheme using deep reinforcement learning in 5g networks. Applied Sciences, 10(12), 2020.

Modifications for RL and results

- Working in the OpenAI Gym mountain car environment
- Preexisting code
  - Changed rewards to be continuous
  - Fixed final time instead of ending iteration once reaching the goal
- Saw limited success from RL
- Rarely if ever found a solution
  - Most training cycles did not converge
- **Shortcoming of RL:** Fragile so we had hard time getting good solutions for OC problem

Suboptimal solution rendered (RL Method)



- Global solution is sub-optimal
- Took many episodes to get the car to the top of the mountain with our given conditions.

Optimal control method (preview)

- We aim estimate the value function

$$\Phi(t, z_t) = \min_u J_{t,\mathbf{z}}[u] \quad \text{subject to} \quad (1),$$

  which satisfies the Hamilton-Jacobi-Bellman PDE

- Our global optimal policy can be obtained through

$$\psi(s, \mathbf{z}) = \arg\max_u \mathcal{H}\left(s, \mathbf{z}, \nabla_z \Phi(s, z), u\right), \tag{3}$$

  where the Hamiltonian is defined as

$$\mathcal{H}(s, \mathbf{z}, p, u) = -p^\top \left(f(s, \mathbf{z}) + \mathbf{B}u\right) - L(s, \mathbf{z}, u).$$

- We will parameterize Φ using a neural network and train using the feedback form (3) and HJB

Summary

Progress:

- Adopted traditional continuous mountain car as control problem
- Found local solutions using numerical ODE solvers and nonlinear optimization
- Implemented actor-critic algorithm and found very suboptimal solutions, if a solution was even found

Next steps:

- Implement optimal control approach
- Optimize reinforcement learning method

Thank you! Any Questions?

References I

Hoang Thi Huong Giang, Tran Nhut Khai Hoan, Pham Duy Thanh, and Insoo Koo.
Hybrid noma/oma-based dynamic power allocation scheme using deep reinforcement learning in 5g networks.
Applied Sciences, 10(12), 2020.

Andrew William Moore.
Efficient memory-based learning for robot control.
Technical report, University of Cambridge, 1990.

U M Ascher and C Greif.
A First Course on Numerical Methods.
SIAM. SIAM, 2011.

R S Sutton and AG Barto.
Reinforcement learning.
The MIT Press. The MIT Press, 2nd edition edition, 2018.

References II

Derek Onken, Levon Nurbekyan, Xingjian Li, Samy Wu Fung, Stanley Osher, and Lars Ruthotto.
A Neural Network Approach for High-Dimensional Optimal Control.
arXiv, 2021.

Amir Beck.
Introduction to Nonlinear Optimization.
SIAM. SIAM, 10 2014.