Learning HJB Viscosity Solutions with PINNs for Optimal Control

Alena Shilova

In collaboration: Thomas Delliaux, Philippe Preux and Bruno Raffin

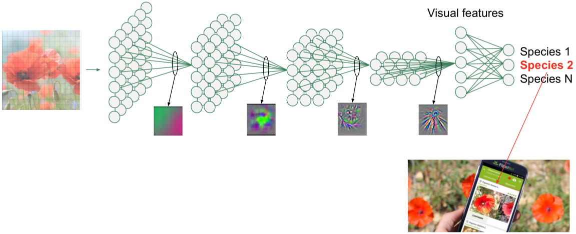
January 17, 2024

Introduction

Shortly about me

- My name is Alena (pronounced Alyona) Shilova
- PostDoc at Inria Scool and DataMove teams
- Before - PhD at Inria HiePACS and Zenith teams
- Even before - Master of Data Science in Skoltech-MIPT, Moscow
- Knowledge in
  - machine learning
  - efficient deep learning (high performance deep learning)
  - reinforcement learning
  - SciML
  - optimization
- Contribution to open source:
  - `rotor`, an optimal rematerialization tool compatible with PyTorch
  - `rlberry`, an RL package for research and education

PhD work



- Part of The HPC-BigData INRIA Project LAB (IPL)
- Work with Pl@ntNet, a platform identifying plants from pictures
- To scale, Pl@ntNet goes to larger models, more species
- Training is thus more time and memory-consuming
- I have focused on solving memory issues
- My contributions are in
  - rematerialization (saving memory by recomputing activations)
  - activation offloading (saving memory by data transfers)
  - pipelined model parallelism

PostDoc work

- Collaboration with Philippe P. (Scool) and Bruno R. (DataMove)
- At first, HPC for RL
  - *e.g.* Asynchronous Advantage Actor-Critic (A3C)
  - *e.g.* learning world models for MBRL
- SciML for RL
  - Use Physics Informed Neural Networks (PINNs) for model learning
  - Even better, use PINNs to learn Hamilton Jacobi Bellman eq. (HJB)
  - HJB is a continuous-time counterpart of Bellman equation

PostDoc work

- Collaboration with Philippe P. (Scool) and Bruno R. (DataMove)
- At first, HPC for RL
    - *e.g.* Asynchronous Advantage Actor-Critic (A3C)
    - *e.g.* learning world models for MBRL
- SciML for RL
    - Use Physics Informed Neural Networks (PINNs) for model learning
    - Even better, use PINNs to learn Hamilton Jacobi Bellman eq. (HJB)
    - HJB is a continuous-time counterpart of Bellman equation

Bridging SciML with Optimal Control!

PostDoc work

- Collaboration with Philippe P. (Scool) and Bruno R. (DataMove)
- At first, HPC for RL
  - *e.g.* Asynchronous Advantage Actor-Critic (A3C)
  - *e.g.* learning world models for MBRL
- SciML for RL
  - Use Physics Informed Neural Networks (PINNs) for model learning
  - Even better, use PINNs to learn Hamilton Jacobi Bellman eq. (HJB)
  - HJB is a continuous-time counterpart of Bellman equation

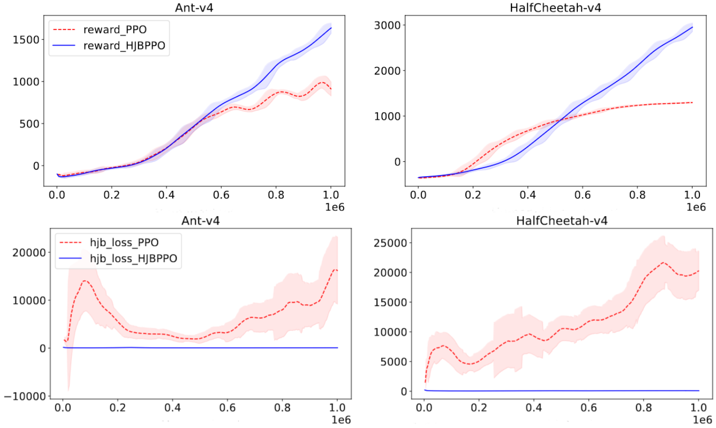
Bridging SciML with Optimal Control!

Promising approach for Continuous Time Reinforcement Learning!

Continuous Time vs Discrete Time RL

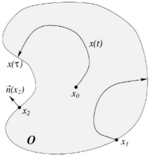
Motivation

- Physical systems and control tasks operate in continuous time
- But most SOTA RL algorithms rely on discrete time assumption



Continuous-time reinforcement learning: problem definition

Continuous time framework

- State space: O, control space: U 

- System dynamics:
  $\frac{dx}{dt} = f(x(t), u(t)) \iff x(t) = x(0) + \int_0^t f(x(t), u(t)) dt$
- Reward function: r defined on $\bar{O}$
- Exit reward function: R defined on ∂O
- Cumulative discounted reward:
  $J(x_0, u(t)) = \int_0^\tau \gamma^t r(x(t), u(t))\, dt + \gamma^\tau R(x(\tau))$, τ is exit time
- **Optimal value function**: $V(x) = \sup_{u(t)} J(x, u(t))$

Hamilton Jacobi Bellman equation

Hamilton Jacobi Bellman

$$V(x)\log\gamma + \sup_{u\in\mathcal{U}}\left[\nabla_x V(x)^T \cdot f(x,u) + r(x,u)\right] = 0 \quad x \in O$$

Optimal control can be obtain by setting

$$\pi(x) \in \arg\sup_{u\in U}\left[\nabla_x V(x)^T \cdot f(x,u) + r(x,u)\right]$$

Challenges

- Value function can be in general non-smooth function
- HJB can have no "strong" solutions, but an infinity of "weak" ones
    - *i.e.* $V \in C(\bar{O})$ but $V \notin C^1(\bar{O})$

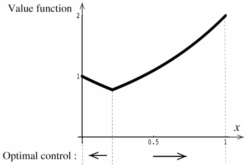
Example (Munos, 2000)

Let $x(t) \in [0,1]$, the control $u(t) \in \{-1,+1\}$ and $\frac{dx}{dt} = u$.
Moreover, $r = 0$ everywhere and $R > 0$ then from the definition on V, we have :

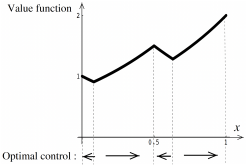
$$V(x) = \max\{R(0)\gamma^x, R(1)\gamma^{1-x}\}$$

and HJB

$$V(x)\log\gamma + \max\{V'_x(x), -V'_x(x)\} = 0$$



Optimal solution for $R(0) = 1$, $R(1) = 2$ and $\gamma = 0.3$



Generalized solution for $R(0) = 1$, $R(1) = 2$ and $\gamma = 0.3$

Viscosity solutions

Viscosity solutions

Let $H(x, W, \nabla W) = -W(x)\log\gamma - \sup_{u\in\mathcal{U}}[\nabla W(x)f(x,u) + r(x,u)]$
Then HJB can be expressed as

$$H(x, W, \nabla W) = 0.$$

Viscosity subsolution

$W \in C(O)$ is a viscosity subsolution of HJB in O if $\forall\phi \in C^1(O)$ and $\forall x \in O$ such that $\phi \geq W$ on O and $\phi(x) = W(x)$, we have:

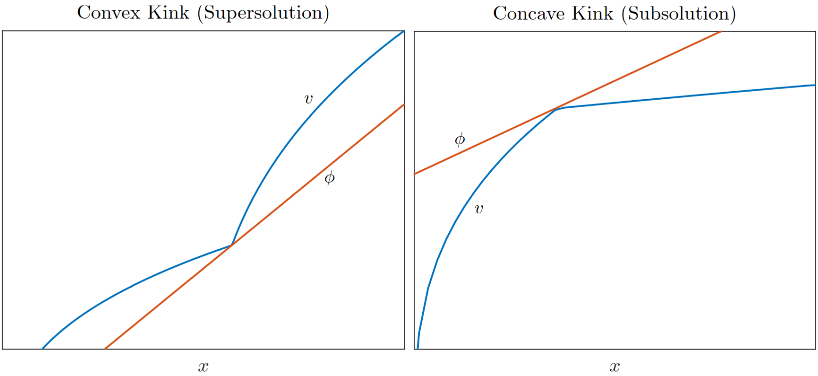
$$H(x, \phi(x), \nabla\phi(x)) \leq 0$$

Viscosity supersolution

$W \in C(O)$ is a viscosity supersolution of HJB in O if $\forall\phi \in C^1(O)$ and $\forall x \in O$ such that $\phi \leq W$ on O and $\phi(x) = W(x)$, we have:

$$H(x, \phi(x), \nabla\phi(x)) \geq 0$$

Viscosity solutions



Viscosity solution

If W is a viscosity subsolution and a supersolution then it is a viscosity solution.

Theorem

Value function $V \in C(\bar{O})$ is the unique viscosity solution of HJB in O .

Viscosity solutions

Intuition behind viscosity solution

If $V \in C^1(O)$:

- Possible to verify the HJB equation for all $x \in O$.
- Verifying the HJB equation is equivalent to be a viscosity solution.

If $V \notin C^1(O)$:

- Impossible to verify the HJB equation where V is not differentiable.
- Alternatively, replace V by $\psi \in C^1(O)$ where it is not differentiable.

Solving HJB equation in practice

Solving HJB equation in practice

Numerical methods (Munos, 2000)

Example: finite difference (FD) and finite element method (FEM):

- Results in solving dynamic programming with value iteration
- Convergence of schemes is guaranteed.
- Can be proved to find viscosity solutions
- One major problem: curse of dimensionality.

Solving with neural networks

Solving PDEs with phisycs-informed neural networks (PINNs):

- Can cope with curse of dimensionality.
- No convergence guarantees.
- Don't take viscosity into account

Solving HJB equation in practice

Numerical methods (Munos, 2000)

Example: finite difference (FD) and finite element method (FEM):

- Results in solving dynamic programming with value iteration
- Convergence of schemes is guaranteed.
- Can be proved to find viscosity solutions
- One major problem: curse of dimensionality.

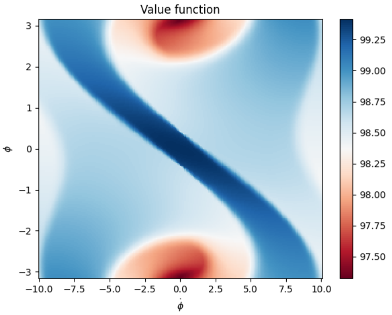
Solving with neural networks

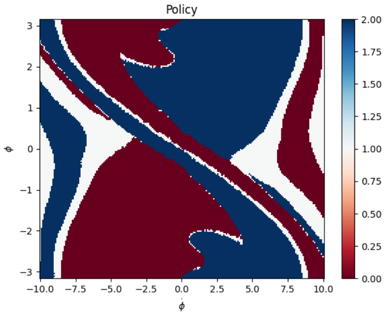
Solving PDEs with phisycs-informed neural networks (PINNs):

- Can cope with curse of dimensionality.
- No convergence guarantees.
- Don't take viscosity into account → our work

Value function and policy for FEM scheme with Value Iteration

- Resolution : 200 by 200.
- Stopping criterion : $\left\| V_n^\delta - V_{n-1}^\delta \right\|_\infty \leq \epsilon$ with $\epsilon = 10^{-5}$.





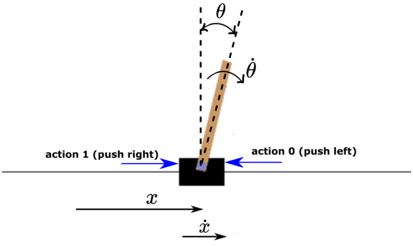
Dynamic programming on pendulum

The main problem of this method is the curse of dimensionality

Example: Cartpole from gym

- State space is a subspace of $\mathbb{R}^4$.
- If we want 32 points per axis, the grid will contain $32^4 = 2^{20}$ states.

$\Rightarrow$ The problem quickly becomes intractable.



Physics-informed neural network method

Data-driven Solutions of Nonlinear PDEs

In order to solve a differential equation

$$\begin{cases} F(x, W(x), \nabla_x W(x), \nabla_x^2 W(x)) = 0, & W : \bar{O} \to \mathbb{R}, x \in O \\ B_k(x, W(x), \nabla_x W(x), \nabla_x^2 W(x)) = 0, & x \in \partial O, k \leq K_1 \\ G_k(x, W(x), \nabla_x W(x), \nabla_x^2 W(x)) \leq 0, & x \in \partial O, k \leq K_2 \end{cases}$$

we design these losses:

- $\mathcal{L}_{PDE}(\theta) = \frac{1}{N_F} \sum_{i=1}^{N_F} \left(F(x_i, W(x_i,\theta), \nabla_x W(x_i,\theta), \nabla_x^2 W(x_i,\theta)) \right)^2$
- $\mathcal{L}_{B_k}(\theta) = \frac{1}{N_B^k} \sum_{i=1}^{N_B^k} \left(B_k(x_i, W(x_i,\theta), \nabla_x W(x_i,\theta), \nabla_x^2 W(x_i,\theta)) \right)^2$
- $\mathcal{L}_{G_{k'}}(\theta) =$
  $\frac{1}{N_G^{k'}} \sum_{i=1}^{N_G^{k'}} \left(\left[G_{k'}(x_i, W(x_i,\theta), \nabla_x W(x_i,\theta), \nabla_x^2 W(x_i,\theta)) \right]^+ \right)^2$

where $[f(x)]^+ = \max\{f(x), 0\}$.

Physics-informed neural networks

One should train a neural network $W(x, \theta)$ that minimizes:

$$\mathcal{L}(\theta) = \mathcal{L}_{PDE}(\theta) + \sum_{k=1}^{K_1} \lambda_k \mathcal{L}_{B_k}(\theta) + \sum_{k=1}^{K_2} \lambda'_k \mathcal{L}_{G_k}(\theta). \tag{1}$$

where $\lambda_k, \lambda'_k > 0$.

Example

$$\begin{cases} W'_x - W = 0 \quad \text{on } O = [0, 1) \\ W(0) = 1 \end{cases} \tag{2}$$

The solution can be found by minimizing the loss

$$\mathcal{L}(\theta) = \|W'_x(., \theta) - W(., \theta)\|_2^2 + \lambda \|W(0, \theta) - 1\|_2^2 \tag{3}$$

The first term is called a PDE loss and the second term a boundary loss.

Physics-informed neural networks and viscosity

Problem with PINNs approach

- How can we ensure that our model approximate the value function, that is, the unique viscosity solution of the HJB equation ?

Viscosity stability lemma

Let $W^\epsilon \in C(O)$ be a viscosity subsolution (resp. a super solution) of

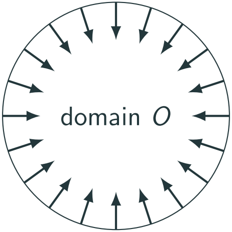
$$W^\epsilon(x) + F^\epsilon(x, W^\epsilon(x), \nabla_x W^\epsilon(x), \nabla_x^2 W^\epsilon(x)) = 0. \tag{4}$$

Suppose that $F^\epsilon \to F$ **uniformly** on every compact subset of O, and $W^\epsilon \to W$ **uniformly** on compact subsets of $\bar{O}$. Then W is a viscosity subsolution (resp. a supersolution) of Eq. (4) for $\epsilon = 0$.

Idea

- Solve $H(x, V^\epsilon(x), \nabla V^\epsilon(x)) = \epsilon \Delta V^\epsilon(x)$ with ϵ decreasing.
- The above equation has a **unique smooth solution**.

Boundary conditions



Boundary condition

How to impose to stay inside O without reaching ∂O?
$\Rightarrow$ By imposing $f(x, u^*(x))^T \eta(x) < 0$ with $\eta(x)$ the external normal vector at $x \in \partial O$.

It is equivalent to

$$-H(x, W, \nabla_x W + \alpha\eta(x)) \leq 0 \quad \forall \alpha \leq 0, x \in \partial O.$$

Losses for HJB PINNs

$$\mathcal{L}_O(\theta, \mathcal{S}_O) = \frac{1}{N_F} \sum_{i=1}^{N_F} \left(H(x_i, W^\epsilon(x_i, \theta), \nabla W^\epsilon(x_i, \theta)) - \epsilon \mathrm{Tr}(\nabla^2 W^\epsilon(x_i, \theta)) \right)^2$$

$$\mathcal{L}_{\partial O}(\theta, \mathcal{S}_{\partial O}) = \frac{1}{N_B} \sum_{i=1}^{N_B} \left([-H(x_i, W^\epsilon(x_i, \theta), \nabla W^\epsilon(x_i, \theta) + \alpha \eta(x_i))]^+ \right)^2$$

MSE regularization loss to encourage uniform convergence:

$$\mathcal{L}_R(\theta, \mathcal{S}_O) = \frac{1}{N_F} \sum_{i=1}^{N_F} \left(W^\epsilon(x_i, \theta) - W^{\epsilon_{n-1}}(x_i, \theta_{\epsilon_{n-1}}) \right)^2 \quad x_i \in \mathcal{S}_O \tag{5}$$

where $W^{\epsilon_{n-1}}(x, \theta_{\epsilon_{n-1}})$ is the best function obtained for ϵ_{n-1}.

The final loss is:

$$\mathcal{L}(\theta, \mathcal{S}_O, \mathcal{S}_{\partial O}) = \mathcal{L}_O(\theta, \mathcal{S}_O) + \lambda \mathcal{L}_{\partial O}(\theta, \mathcal{S}_{\partial O}) + \lambda_R \mathcal{L}_R(\theta, \mathcal{S}_O). \tag{6}$$

We use **uniform sampling** to get samples from O

For now, we assume that $r(x, u)$ and $f(x, u)$ are **known**.

How to schedule ϵ?

Non-adaptive scheduler

$$\epsilon_{n+1} = \epsilon_n k_\epsilon \quad \text{if } n+1 \equiv 0 \mod N_u \quad \text{otherwise, } \epsilon_n$$

Adaptive scheduler

$$\epsilon_{n+1} = \begin{cases} \frac{k_\epsilon \delta(\epsilon_n, \theta_{n-1})}{\delta(\epsilon_n, \theta_n)} \epsilon_n & \text{if } k_\epsilon \delta(\epsilon_n, \theta_{n-1}) \leq \delta(\epsilon_n, \theta_n), \\ & \text{and } \mathcal{L}(\theta_i) \geq \mathcal{L}(\theta_{i-1}) \ \forall i : n - n_\epsilon + 1 \leq i \leq n \\ \epsilon_n & \text{otherwise,} \end{cases}$$

where $\delta(\epsilon, \theta) = \frac{1}{N_F} \sum_i \left\| \epsilon \mathrm{Tr}(\nabla_x^2 W^\epsilon(x_i, \theta)) \right\|^2$

Hybrid scheduler

- Start from high enough ϵ_0 to improve stability.
- Do a given number of ϵ updates with the non-adaptive scheduler.
- Switch to the adaptive scheduler until the end of the training phase.

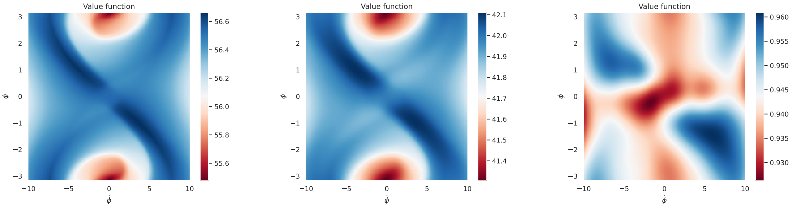
Physics-informed neural networks. Non-Adaptive Scheduler

Results with non-adaptive scheduler

We used

- $k_\epsilon = 0.5$ with $N_u = 10$ (left)
- $k_\epsilon = 0.5$ with $N_u = 25$ (middle)
- $k_\epsilon = 0.5$ with $N_u = 75$ (right).

The non-adaptive scheduler doesn't control under-/over-fitting



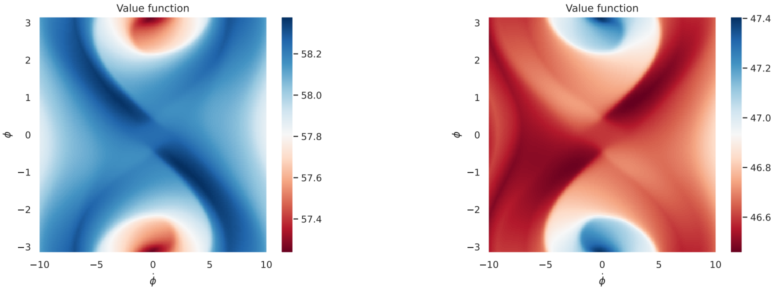
Physics-informed neural networks. Adaptive Scheduler

Results with adaptive scheduler

We used $k_\epsilon = 0.9$ with $\epsilon_0 = 1$ (left) and $\epsilon_0 = 10^{-3}$ (right).

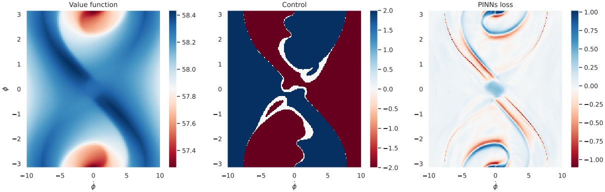
- The adaptive scheduler can converge but **slowly**.
- Starting from a small ϵ_0 increases the speed but can **diverge**.

Adaptive scheduler may be unstable if starting from small ϵ



Physics-informed neural networks. Hybrid Scheduler

It keeps the strengths of both schedulers without their weaknesses!



Names	Hyperparameters	values
number of sampled points	N_D	200000
batch size	N_S	100
learning rate	ν	0.00085
patience adaptive scheduler	n_ϵ	7
boundary loss coefficient	λ	10^{-1}
reg loss coefficient	λ_R	10^{-3}
starting ϵ	ϵ_0	1
number of epochs between ϵ updates	N_u	10
non-adaptive scheduler coefficient	k_ϵ	0.1
adaptive scheduler coefficient	$k_\epsilon^{'}$	0.99
number of ϵ updates with non-adaptive scheduler	N_ϵ	5

Physics-informed neural networks

Cumulative rewards for the different methods

Problem	Method	N	Mean	Std
Pendulum	FEM (VI)	200	4133.71	433.02
	A2C	NA	2180.22	766.25
	PPO	NA	3273.51	906.41
	PINNs	NA	3809.59	542.50
CartPole	A2C	NA	1697.15	398.81
	PPO	NA	5000.0	0.0
	PINNs	NA	5000.0	0.0
CartPole Swing-Up	A2C	NA	90.87	0.73
	PPO	NA	970.63	130.3
	PINNs	NA	723.3	175.16
Acrobot	PPO	NA	1387.3	294.1
	PINNs	NA	506.4	180.8

Conclusion

- Continuous-Time RL depends on solving HJB equation
- Solving it in general case is a **challenging** task
- Finding viscosity solutions is even harder
- It can be solved
  - Either with **numerical scheme** (FD, FEM, . . .), but limited scalability
  - Or with **neural networks** (PINNs-like approach), but less guarantees
- We use ϵ-**scheduling** scheme + **PINNs** to find HJB viscosity solutions
- Works for classical control problems

Conclusion

- Continuous-Time RL depends on solving HJB equation
- Solving it in general case is a **challenging** task
- Finding viscosity solutions is even harder
- It can be solved
  - Either with **numerical scheme** (FD, FEM, ...), but limited scalability
  - Or with **neural networks** (PINNs-like approach), but less guarantees
- We use ϵ**-scheduling** scheme + **PINNs** to find HJB viscosity solutions
- Works for classical control problems
- $\Rightarrow$ needs to be adapted for more complicated environments

Future perspectives

- Continue to work at the crossroads of SciML, RL, optimal control
  - Improving PINNs solver for the HJB equation
    - Consider adaptive sampling without breaking viscosity
    - Analyze the effect of ϵ-scheduling on convergence to V
    - Scale it to more complex environments
  - Using PINNs, Neural Operators to learn model in MBRL
    - PINNs when the model is partially known
    - Neural ODE/Neural Operators when it is fully unknown
- Use HPC expertise to further scale the algorithms
  - Use data/model parallelisms to train larger DNNs for the above tasks
  - Memory saving strategies for SciML forward-backward graphs

Other contributions

rlberry

- Python library that manages RL experiments
- Active contributor

AdaStop

Sequential testing for efficient and reliable comparisons of RL Agents.

- It combines Permutation testing

+ Group Sequential testing

+ Multiple Hypothesis testing

MiCaRL

Entropy regularized RL with Cascading networks.

- Use Politex update (Abbasi-Yadkori et al., 2019) for Policy Iteration
- Requires summing different Q-value neural networks
- Can be done efficiently with Cascading networks