

Introduction to reinforcement learning for optimal control

Theilo Terrisse

FAU Erlangen-Nürnberg
Chair for Dynamics, Control, Machine Learning and Numerics
Alexander von Humboldt Professorship

Under the supervision of Prof. E. Zuazua and N. De Nitti

July 7th 2023

Goals of this Presentation

- Give an introduction to reinforcement learning (RL), starting from optimal control theory.
- Illustrate how RL can be considered to cope with two limitations of optimal control:
 - the curse of dimensionality;
 - the difficulty of model identification.

- 1 Optimal control and its limitations
- 2 From optimal control to reinforcement learning
 - Dynamic programming
 - Q-learning
- 3 Implementations of RL
 - Cart-pushing problem
 - Controlling the viscous Burgers equations
- 4 Broadening the scope of RL
- 5 Conclusion

Outline

- 1 Optimal control and its limitations
- 2 From optimal control to reinforcement learning
 - Dynamic programming
 - Q-learning
- 3 Implementations of RL
 - Cart-pushing problem
 - Controlling the viscous Burgers equations
- 4 Broadening the scope of RL
- 5 Conclusion

Optimal control problems

Definition

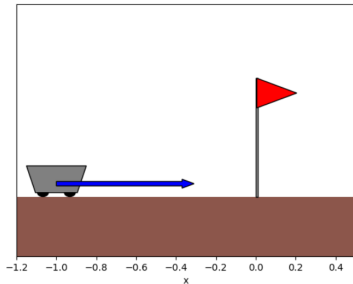
Let $T > 0$ and $x_0 \in X$. In an optimal control problem, we want to

$$\begin{aligned} \underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} \quad & J(\mathbf{x}_u, \mathbf{u}) = C_f(\mathbf{x}_u(T)) + \int_0^T C(\mathbf{x}_u(t), \mathbf{u}(t)) dt \\ \text{s.t.} \quad & \begin{cases} \dot{\mathbf{x}}_u(t) = f(t, \mathbf{x}_u(t), \mathbf{u}(t)) & \forall t \in [0, T], \\ \mathbf{x}_u(0) = x_0 \end{cases} \end{aligned}$$

where $J : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$. A minimizer of J is called an *optimal control*.

Example:

$$\begin{aligned} \underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} \quad & \lambda_P \mathbf{x}_u(T)^2 + \lambda_V \dot{\mathbf{x}}_u(T)^2 + \int_0^T \mathbf{u}(t)^2 dt \\ \text{s.t.} \quad & \begin{cases} m\ddot{\mathbf{x}}_u(t) = \mathbf{u}(t) & \forall t \in [0, T], \\ \mathbf{x}_u(0) = x_0, \\ \dot{\mathbf{x}}_u(0) = 0 \end{cases} \end{aligned}$$



Optimal control problems

Definition

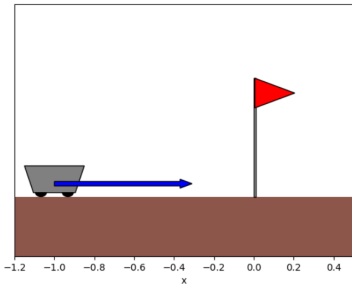
Let $T > 0$ and $x_0 \in X$. In an optimal control problem, we want to

$$\begin{aligned} \underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} \quad & J(\mathbf{x}_u, \mathbf{u}) = C_f(\mathbf{x}_u(T)) + \int_0^T C(\mathbf{x}_u(t), \mathbf{u}(t)) dt \\ \text{s.t.} \quad & \begin{cases} \dot{\mathbf{x}}_u(t) = f(t, \mathbf{x}_u(t), \mathbf{u}(t)) & \forall t \in [0, T], \\ \mathbf{x}_u(0) = x_0 \end{cases} \end{aligned}$$

where $J : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$. A minimizer of J is called an *optimal control*.

Example:

$$\begin{aligned} \underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} \quad & \lambda_P \mathbf{x}_u(T)^2 + \lambda_V \dot{\mathbf{x}}_u(T)^2 + \int_0^T \mathbf{u}(t)^2 dt \\ \text{s.t.} \quad & \begin{cases} m\ddot{\mathbf{x}}_u(t) = \mathbf{u}(t) & \forall t \in [0, T], \\ \mathbf{x}_u(0) = x_0, \\ \dot{\mathbf{x}}_u(0) = 0 \end{cases} \end{aligned}$$



⚠ Real systems are non-linear!

Optimal control: numerical toolbox

Methods derived from *necessary conditions* of optimality

Many options, among which:

- Adjoint-methodology: express the gradient $\nabla_{\mathbf{u}} J$ in terms of the co-state.
- Project \mathbf{u} on a space of finite dimension and use non-linear programming.

→ No guarantee of finding an optimal solution.

Methods derived from *sufficient conditions* of optimality

Use Hamilton-Jacobi-Bellman equations to find the optimal *value function* V^* , and derive a *feed-back* control from it:

$$\begin{cases} \partial_t V^* + \min_{u \in U} \{C(x, u) + \langle f(x, u), \nabla V^* \rangle\} = 0 & \forall (x, t) \\ V^*(x, T) = C_f(x) & \forall x \end{cases} \quad (1)$$

→ Suffers from the *curse of dimensionality*.

Optimal control: numerical toolbox

Methods derived from *necessary conditions* of optimality

Many options, among which:

- Adjoint-methodology: express the gradient $\nabla_{\mathbf{u}} J$ in terms of the co-state.
- Project \mathbf{u} on a space of finite dimension and use non-linear programming.

→ No guarantee of finding an optimal solution.

Methods derived from *sufficient conditions* of optimality

Use Hamilton-Jacobi-Bellman equations to find the optimal *value function* V^* , and derive a *feed-back* control from it:

$$\begin{cases} \partial_t V^* + \min_{u \in U} \{C(x, u) + \langle f(x, u), \nabla V^* \rangle\} = 0 & \forall (x, t) \\ V^*(x, T) = C_f(x) & \forall x \end{cases} \quad (1)$$

→ Suffers from the *curse of dimensionality*.

+ All these methods require an explicit mathematical model!

Outline

- 1 Optimal control and its limitations
- 2 From optimal control to reinforcement learning
 - Dynamic programming
 - Q-learning
- 3 Implementations of RL
 - Cart-pushing problem
 - Controlling the viscous Burgers equations
- 4 Broadening the scope of RL
- 5 Conclusion

Reinforcement learning: A first definition

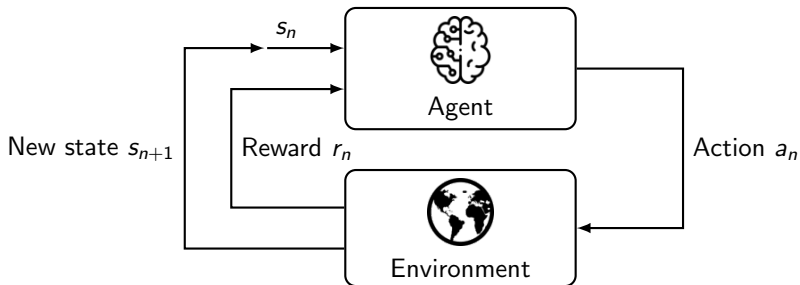


Figure: The agent–environment interaction [SB18, Adapted from Fig. 3.1].

→ The agent aims to maximize the cumulative reward $\sum_{n=0}^{N-1} r_n + r_f$.

Reinforcement learning: A first definition



Interact with the system without knowing the model.



Many possible games (about 10^{600}).
AlphaGo beat professional *Go* players in 2016.

Discrete dynamical system

Discrete dynamics

$$\begin{aligned} & \underset{\pi}{\text{maximize}} && \sum_{n=0}^{N-1} r_n(s_n, a_n) + r_f(s_N) \\ & \text{s.t.} && s_{n+1} = f_n(s_n, \pi_n(s_n)) \end{aligned}$$

→ s_{n+1} is assumed to depend only on s_n and a_n .

Policy π

Function that maps states to actions: behaviour of the agent.

Discrete value function

$$\begin{cases} V_n^\pi(s) = \sum_{k=n}^{N-1} r_k(s_k, a_k) + r_f(s_N) \\ s_n = s \end{cases}$$

Value function

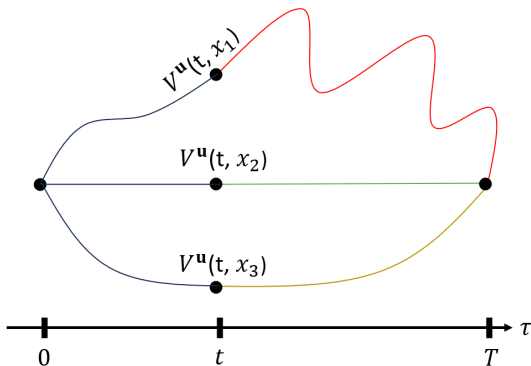


Figure: Representation of the value function.

Bellman equation

Hamilton-Jacobi-Bellman equation

$$\partial_t V^* + \min_{u \in U} \{C(x, u) + \langle f(x, u), \nabla V^* \rangle\} = 0$$



Bellman equation

$$V_n^*(s) = \max_{a \in A} \{r_n(s, a) + V_{n+1}^*(f_n(s, a))\}$$

Deriving a policy

If the optimal value-function V^* is known, the optimal policy π^* can be derived as its *greedy policy*:

$$\pi^*(s) = \arg \max_{a \in A} \{r(s, a) + V_n^*(f_n(s, a))\}$$

Curse of dimensionality

The complexity of the Bellman recursion algorithm is

$$O(N_T \times N_X^{\dim(X)} \times N_A^{\dim(A)}),$$

with N_T , N_X and N_A the number of discretization steps in each direction of time, state and action respectively.

Value iteration

Value iteration

Initialize some function $V^0 : S \times A$. For $k \in \mathbb{N}$:

$$V_n^{k+1}(s) = \max_{a \in A} \{r_n(s, a) + V_{n+1}^k(f_n(s, a))\} \quad \forall n, s$$

Then, $V_n^k \xrightarrow[k \rightarrow \infty]{} V_n^*$

Value iteration

Value iteration

Initialize some function $V^0 : S \times A$. For $k \in \mathbb{N}$:

$$V_n^{k+1}(s) = \max_{a \in A} \{r_n(s, a) + V_{n+1}^k(f_n(s, a))\} \quad \forall n, s$$

Then, $V_n^k \xrightarrow[k \rightarrow \infty]{} V_n^*$

→ This convergence result also holds for the *in-place* value iteration algorithm, *i.e.*

$$\begin{cases} V_n^{k+1}(s) = \max_{a \in A} \{r_n(s, a) + V_{n+1}^k(f_n(s, a))\} & \text{for a given } (n, s) \\ V^{k+1} = V^k & \text{elsewhere} \end{cases}$$

as long as all states and time steps are visited an infinite number of times.

Value iteration

N.B.: The order in which states are visited matters:

- some orders achieve *faster convergence* (the best order being Bellman recursion);
- some states *might not need to be visited* (think of a maze with only one entrance);
- a natural order is the one arising when *interacting with the system*.

Introducing Q

Idea: Replace a value-function of state by a value-function of state *and* action:

Q-function

$$Q_n^\pi(s, a) = r_n(s, a) + \sum_{k=n+1}^{N-1} r_k(s_k, \pi(s_k)) + r_f(s_N)$$

Introducing Q

Idea: Replace a value-function of state by a value-function of state *and* action:

Q-function

$$Q_n^\pi(s, a) = r_n(s, a) + \sum_{k=n+1}^{N-1} r_k(s_k, \pi(s_k)) + r_f(s_N)$$

Why introduce Q? The Bellman equation rewrites:

Bellman equation

$$Q_n^*(s, a) = r_n(s, a) + \max_{a' \in A} \{Q_{n+1}^*(s', a')\}$$

→ Deriving a policy does not require a model and it is cheaper:

$$\pi^*(s) = \arg \max_a (Q^*(s, a)) \quad (2)$$

Q-learning algorithm

Algorithm Q-learning [Wat89]

Data: Exploration factor $\varepsilon > 0$, learning rate $\alpha > 0$, episodes N_{episodes} , horizon $T > 0$.

$\forall t \in \{0, \dots, T-1\}$, $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$, initialize $Q_t(s, a)$ arbitrarily.

for episode = 1, 2, ..., N_{episodes} **do**

Initialize s .

for $t = 0, 1, \dots, T-1$ **do**

$a \leftarrow \pi^\varepsilon(s)$. ← choose action with off-policy

$s' \leftarrow f(s, a)$.

if $t = T-1$ **then** (value-iteration update)

$$Q_{T-1}(s, a) \leftarrow (1 - \alpha)Q_T(s, a) + \alpha [r_{T-1}(s, a) + r_f(s)].$$

else (value-iteration update)

$$Q_t(s, a) \leftarrow (1 - \alpha)Q_t(s, a) + \alpha \left[r_t(s, a) + \max_{a' \in \mathcal{A}} Q_{t+1}(s, a') \right].$$

end if

$s \leftarrow s'$.

end for

end for

Q-learning algorithm

Remarks

- About the parameters of Q-learning:
 - ε sets the amount of exploration:

$$\pi^\varepsilon(s, t) = \begin{cases} \arg \max_{a \in A} Q_t(s, a) & \text{with probability } 1 - \varepsilon \\ a \text{ chosen randomly in } A & \text{with probability } \varepsilon. \end{cases}$$

- The learning rate α characterizes how fast we update Q .
- They set the *exploration-exploitation* balancing of Q-learning.

Q-learning algorithm

Remarks

- About the parameters of Q-learning:
 - ε sets the amount of exploration:

$$\pi^\varepsilon(s, t) = \begin{cases} \arg \max_{a \in A} Q_t(s, a) & \text{with probability } 1 - \varepsilon \\ a \text{ chosen randomly in } A & \text{with probability } \varepsilon. \end{cases}$$

- The learning rate α characterizes how fast we update Q .
 → They set the *exploration-exploitation* balancing of Q-learning.
- Q-learning draws inspiration from temporal difference.

$$Q_t(s, a) \leftarrow Q_t(s, a) + \alpha \underbrace{\left[r_t(s, a) + \overbrace{\max_{a' \in A} Q_{t+1}(s, a')}^{\text{target}} - Q_t(s, a) \right]}_{\text{temporal difference}}$$

Reinforcement learning: recap

(Value-based) reinforcement learning usually involves the following concepts:

- It starts from an arbitrary behaviour (*i.e.* policy).
- It *evaluates* this behaviour, for example using a value function.
- It uses this evaluation to *improve* the behaviour.

This is typically done using some variant of value iteration performed through an interactive exploration of the system.

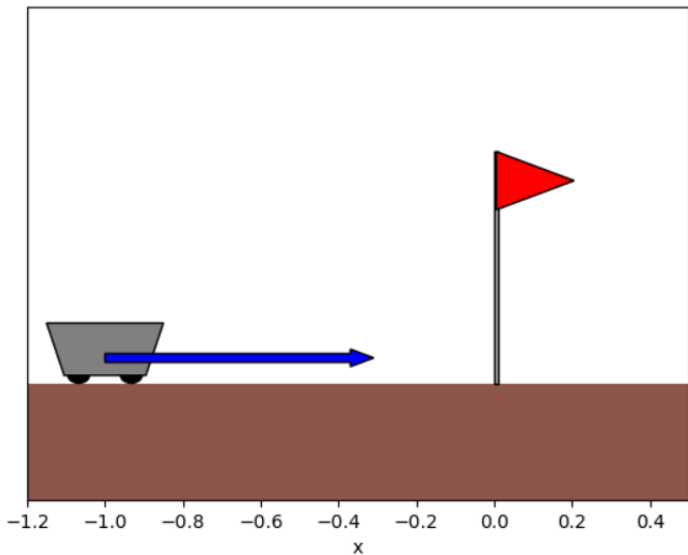
Outline

- 1 Optimal control and its limitations
- 2 From optimal control to reinforcement learning
 - Dynamic programming
 - Q-learning
- 3 Implementations of RL**
 - Cart-pushing problem
 - Controlling the viscous Burgers equations
- 4 Broadening the scope of RL
- 5 Conclusion

Goal of the implementations

- Illustrate and visualize Q -learning to better understand its parameters.
- See how RL applies to a continuous-time optimal control problem.
- Discuss RL from the perspective of the limitations of optimal control we have mentioned:
 - avoid model identification;
 - face the curse of dimensionality through exploration-exploitation balancing;
 - tackle hard non-linear problems.

Cart-pushing problem



Cart-pushing problem

Formulation

Sped-up cart version

Here, $\mathbf{x} : [0, T] \rightarrow \mathbb{R}$ and $\mathbf{u} \in \mathcal{U} = L^2([0, T], U)$. We want to

$$\begin{aligned} & \underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} && \int_0^T (\lambda \mathbf{x}_{\mathbf{u}}(t)^2 + \mathbf{u}(t)^2) dt, \\ & \text{s.t.} && \begin{cases} \dot{\mathbf{x}}_{\mathbf{u}}(t) = \mathbf{u}(t) & \forall t \in [0, T], \\ \mathbf{x}_{\mathbf{u}}(0) = x_0, \end{cases} \end{aligned}$$

where $\lambda > 0$ and x_0 is taken randomly in $[-1, 0]$.

We fix $T = 1$ and $\lambda = 10$.

Sped-up cart

Discretization

We arbitrarily confine action values in $\tilde{U} = [U_L, U_R]$ (with $U_L \in \mathbb{Z}^-$ and $U_R \in \mathbb{N}$), and we introduce discretization steps $\Delta t = \frac{1}{N}$, $\Delta u = \frac{1}{N_U}$ and $\Delta x = \frac{1}{N_X}$ for time, action and state spaces respectively. We look for \mathbf{u} in

$$\tilde{u}_{\Delta t, \Delta u} = \left\{ \sum_{n=0}^{N-1} u_n \mathbb{1}_{[t_n, t_{n+1})} \mid \forall n \in \{0, \dots, N-1\}, u_n \in \tilde{U}_{\Delta u} \right\}$$

where

$$\tilde{U}_{\Delta u} = \{U_L, U_L + \Delta u, \dots, U_R\}$$

and the position lives in

$$\tilde{X}_{\Delta x} = \{X_L, X_L + \Delta u \Delta t, \dots, X_R\}.$$

Then the solution \mathbf{x} to the ODE verifies:

$$x_{n+1} = \mathcal{P}(x_n, u_n)$$

where

$$\begin{aligned} \mathcal{P} : X \times U &\rightarrow X \\ (x, u) &\mapsto x + \Delta t u. \end{aligned}$$

will be called the “simulator”.

Sped-up cart

Discretization

The optimal control problem also rewrites:

$$\underset{u \in \tilde{\mathcal{U}}_{\Delta t, \Delta u}}{\text{minimize}} \lambda \sum_{n=0}^{N-1} \left(\Delta t x_n^2 + \Delta t^2 x_n u_n + \frac{\Delta t^3}{3} u_n^2 \right) + \Delta t \sum_{n=0}^{N-1} u_n^2.$$

Sped-up cart

Discretization

λ	T	N	X_L	X_R	N_X	U_L	U_R	N_U
10	1	20	-9	8	100	-8	8	5

Table: Numerical parameters for the sped up cart problem

Sped-up cart

RL formulation

Formulation as an RL problem

Using the discretization, we formulate the problem in an RL framing:

- $S = \tilde{X}_{\Delta x}$.
- $\mathcal{A} = \tilde{U}_{\Delta u}$.
- The dynamics of the system is $f = \mathcal{P}$.
- $\forall n \in \{0, \dots, N-1\}, \forall s \in S, \forall a \in \mathcal{A}$,

$$r_n(s, a) := r(s, a) = -\lambda \left(\Delta t s_n^2 + \Delta t^2 s_n a_n + \frac{\Delta t^3}{3} a_n^2 \right) - \Delta t a_n^2.$$
- $r_f(x) = 0$.

Sped-up cart

$\varepsilon = 0.85$ and $\alpha = 0.2$

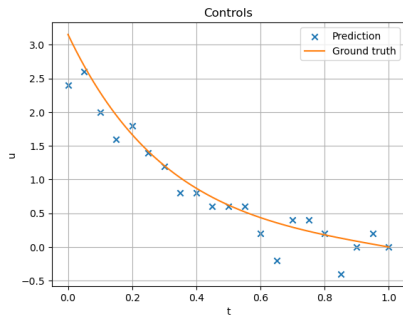
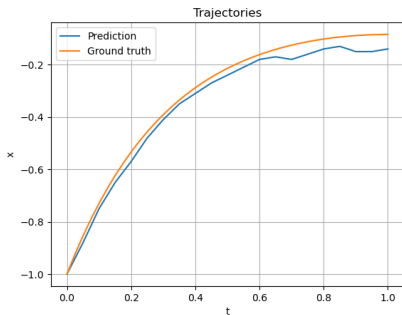


Figure: Trajectory and control for $\varepsilon = 0.85$ and $\alpha = 0.2$, for $x_0 = -1$

The RL agent achieves a total cost 2% greater than the analytical solution.

Sped-up cart

$\varepsilon = 0.85$ and $\alpha = 0.2$

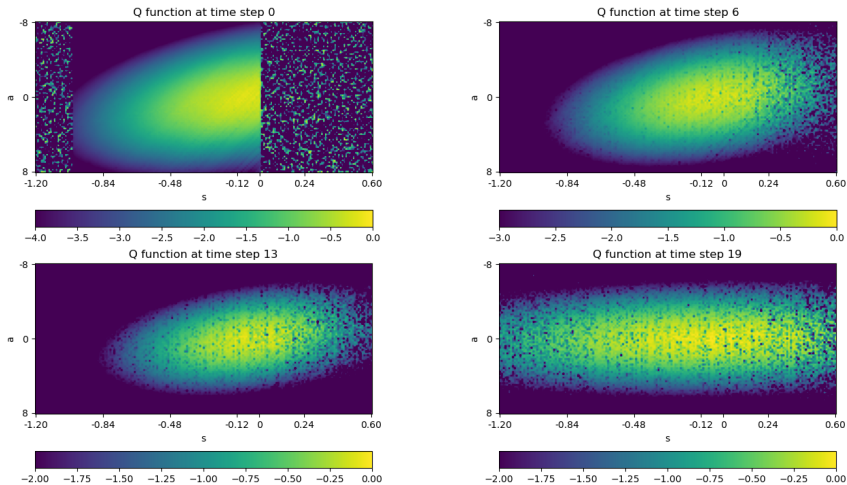


Figure: Q for various time-steps, for $\varepsilon = 0.85$ and $\alpha = 0.2$

Sped-up cart

$\varepsilon = 0.85$ and $\alpha = 0.2$

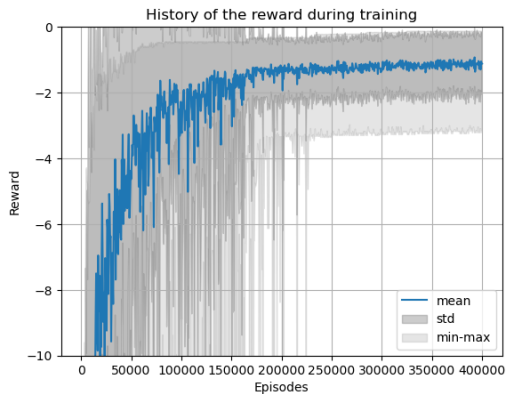


Figure: Training curve for $\varepsilon = 0.85$ and $\alpha = 0.2$

Sped-up cart

$\varepsilon = 0.1$ and $\alpha = 1$

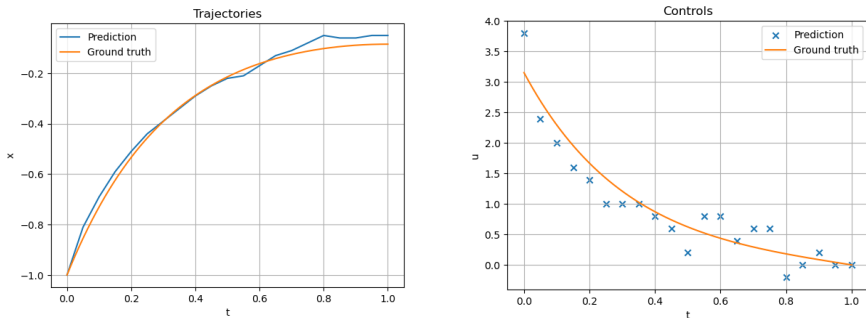


Figure: Trajectory and control for $\varepsilon = 0.1$ and $\alpha = 1$, for $x_0 = -1$

The RL agent achieves a total cost 2% greater than the analytical solution.

Sped-up cart

$\varepsilon = 0.1$ and $\alpha = 1$

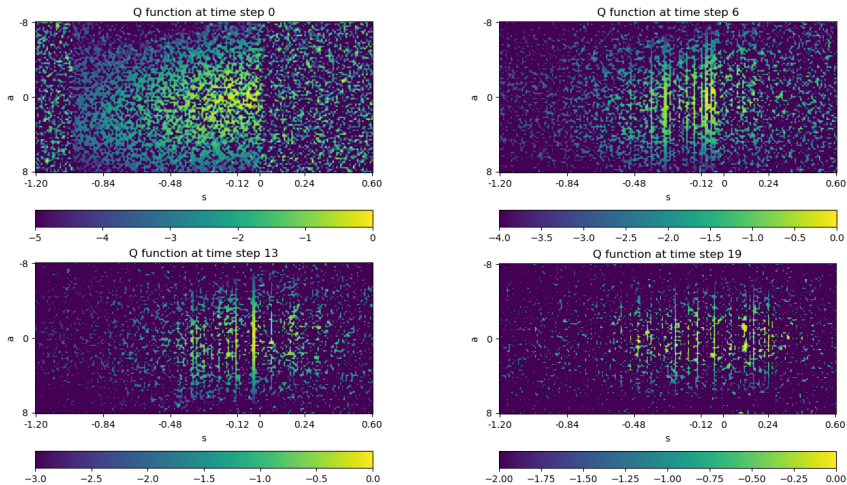


Figure: Q for various time-steps, for $\varepsilon = 0.1$ and $\alpha = 1$

Sped-up cart

$\varepsilon = 0.1$ and $\alpha = 1$



Figure: Training curve for $\varepsilon = 0.1$ and $\alpha = 1$

Sped-up cart

Discussion

Parameters of Q-learning allow to solve the exploration-exploitation dilemma:

- Greater ε lead to more exploration, but also to more variance; hence a slower training process.
- Greater γ leads to steeper updates; in the deterministic case, $\gamma = 1$ is equivalent to value iteration.

Cart-pushing problem

Accelerated cart version

A different algorithm, *proximal policy iteration* [SWD⁺17], is used to solve the full-fledged version of this problem:

$$\begin{aligned} & \underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} && \lambda_P \mathbf{x}_{\mathbf{u}}(T)^2 + \lambda_V \dot{\mathbf{x}}_{\mathbf{u}}(T)^2 + \int_0^T \mathbf{u}(t)^2 dt \\ & \text{s.t.} && \begin{cases} m \ddot{\mathbf{x}}_{\mathbf{u}}(t) = \mathbf{u}(t) & \forall t \in [0, T], \\ \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\ \dot{\mathbf{x}}_{\mathbf{u}}(0) = 0 \end{cases} \end{aligned}$$

where m is considered unknown.

m	λ_P	λ_V	T	N
1	100	50	1	30

Table: Numerical parameters for the accelerated cart problem

Accelerated cart

Solving with PPO

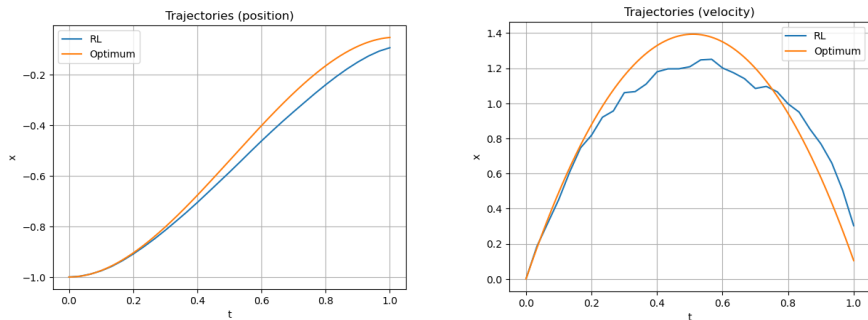


Figure: Trajectory and control given by the trained PPO agent for $x_0 = -1$

The RL agent achieves a cost 21% greater than the optimal.

Accelerated cart

Solving with PPO

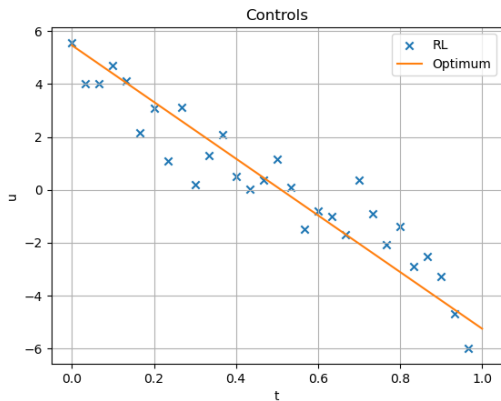


Figure: Control given by the trained PPO agent for $x_0 = -1$

Accelerated cart

Using reinforcement learning

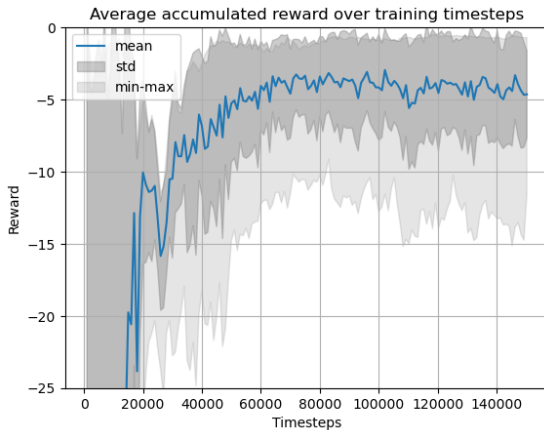


Figure: Training curve for the PPO agent

Accelerated cart

Discussion

- *The model was never given to the RL agent, thus bypassing model identification.*
- The RL agent learns to control the system, but it only gets close to the optimum for this simple problem.
- The RL agent required (optimistically) 90 seconds and 50 000 samples (*i.e.* state transitions) to train. This illustrates the *sample inefficiency* of RL. This should be compared with a model-identification procedure on harder problems.

Towards non-linear problems

Controlling (viscous) Burgers equations

$$\begin{aligned} & \underset{a \in \mathcal{A}}{\text{minimize}} && \beta \|u(T, \cdot) - u^*(\cdot)\|_{L^2(\Omega)}^2 + \int_0^T \|a(t, \cdot)\|_{L^2(\Omega)}^2 dt \\ & \text{s.t.} && \begin{cases} \partial_t u + \partial_x \left(\frac{u^2}{2} \right) - \nu \partial_{xx}^2 u = a(t, x), & t \in (0, T), x \in \Omega, \\ u(0, x) = u^0(x), & x \in \Omega, \\ u(t, x) = 0, & t \in (0, T), x \in \partial\Omega, \end{cases} \end{aligned}$$

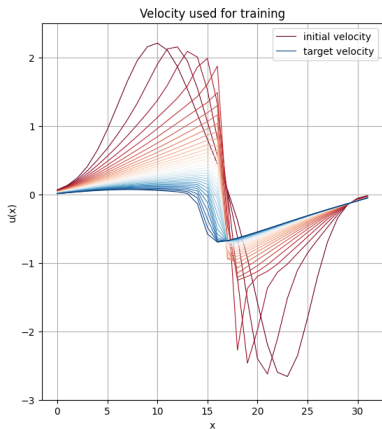
where

$$\mathcal{A} = \left\{ a \in L^2((0, T) \times \Omega) \mid a_L \leq a(t, x) \leq a_U \text{ a.e. in } (0, T) \times \Omega \right\},$$

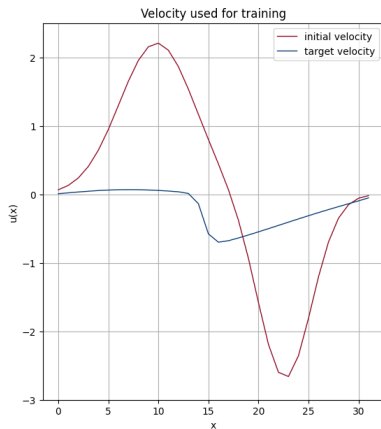
where $0 < a_L < a_U$ are two real numbers.

This problem is tackled starting from code in [THM⁺21, Chapter 16].

Towards non-linear problems



(a) Instance generation process



(b) Initial and target velocities

Figure: Example of an instance of the problem.

Towards non-linear problems

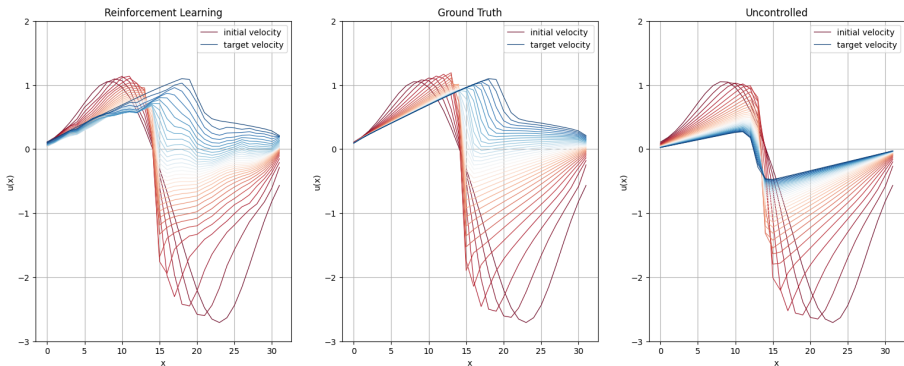


Figure: Performance of the trained agent.

Outline

- 1 Optimal control and its limitations
- 2 From optimal control to reinforcement learning
 - Dynamic programming
 - Q-learning
- 3 Implementations of RL
 - Cart-pushing problem
 - Controlling the viscous Burgers equations
- 4 Broadening the scope of RL
- 5 Conclusion

Classification of methods

→ What mathematical objects are we learning?

- A value-function; different ways of approximating:
 - dynamic programming: value/policy iteration;
 - Monte-Carlo methods (zero-bias by law of large numbers);
 - temporal-difference (low-variance thanks to bootstrapping).
- A policy (without the value function): parameterize π and use the *log-derivative trick* to do gradient ascent.
- Both: *actor-critic* methods.

Classification of methods

→ Are we using a model?

- Model-free methods, *e.g.* using Q .
- Model-based methods: allow planning and data augmentation.

→ Are we using a different policy during training and exploitation phases?

- On-policy algorithms; typically better for early performance.
- Off-policy algorithms; typically, the off-policy is chosen for exploration.

→ Are we using deep neural networks ?

- To learn a model.
- To parameterize a value function.
- To parameterize a policy.

Outline

- 1 Optimal control and its limitations
- 2 From optimal control to reinforcement learning
 - Dynamic programming
 - Q-learning
- 3 Implementations of RL
 - Cart-pushing problem
 - Controlling the viscous Burgers equations
- 4 Broadening the scope of RL
- 5 Conclusion

Conclusion

We introduced reinforcement learning and highlighted some of its useful features:

- *exploration-exploitation balancing* to circumvent the curse of dimensionality;
- working *model-free* and in a *data-driven* manner.

We also noticed some drawbacks of reinforcement learning:

- Difficult *parameter-tuning*;
- *Sample inefficacy*, making a simulator often unavoidable.

Next steps

Further investigation should be carried out regarding hard non-linear problems.





The previous experiments can also be reproduced with noisy dynamics to study RL for stochastic systems.

Lastly, we only scratched the surface of reinforcement learning.

Other active research fields include:

- *Transfer learning*: use a pre-trained model or expert observations to guide and accelerate the training of an agent.
- *Inverse reinforcement learning*: design a reward function from expert observations.
- *Partially-observed states*.
- ...

References

-  Richard S. Sutton and Andrew G. Barto.
Reinforcement learning: An introduction, 2nd ed.
The MIT Press, Cambridge, MA, US, 2018.
-  John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov.
Proximal Policy Optimization algorithms.
ArXiv:1707.06347, 2017.
-  Nils Thuerey, Philipp Holl, Maximilian Mueller, Patrick Schnell, Felix Trost, and Kiwon Um.
Physics-based Deep Learning.
<https://physicsbaseddeeplearning.org>, 2021.
-  C. Watkins.
Learning from delayed rewards.
1989.