From Optimal Control to Reinforcement Learning

Daniel López Montero, Max Golightly

May 2026

Day 1: Optimal Control and Classical Reinforcement Learning

A bit of history: learning from consequences

In 1898, Edward Thorndike formulated the *law of effect*, which states that responses followed by satisfying consequences become more likely, while responses followed by discomfort become less likely.

The Russian physiologist Ivan Pavlov showed in the early 1900s that dogs can be classically conditioned to salivate in response to stimuli.

The American psychologist B.F. Skinner developed the experimental analysis of operant conditioning in the 1930s, in which voluntary behavior is shaped by reinforcement and punishment.

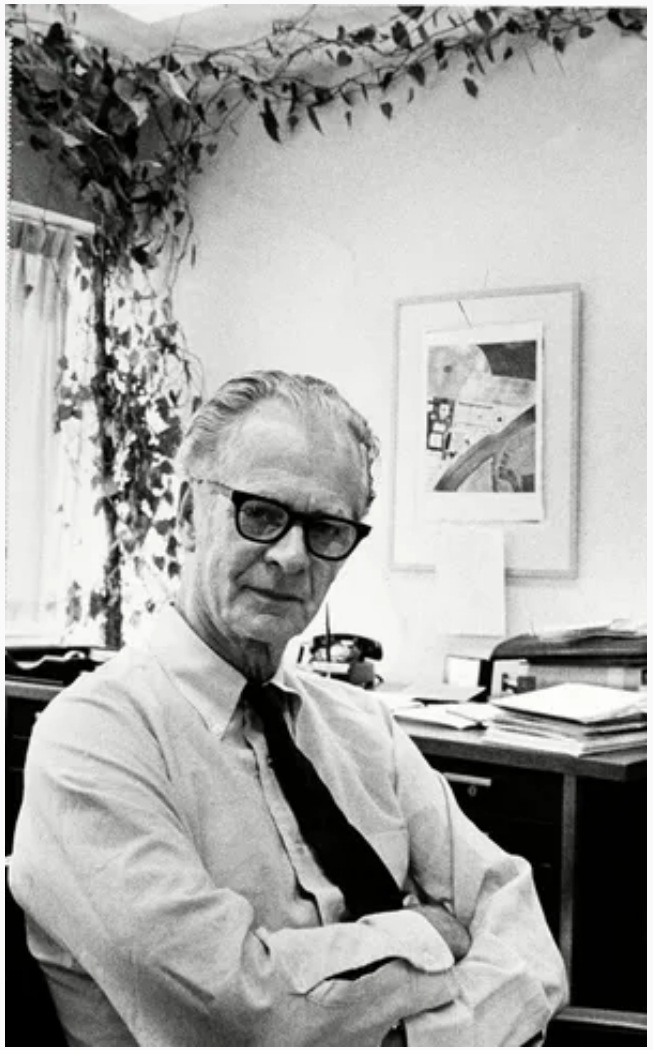
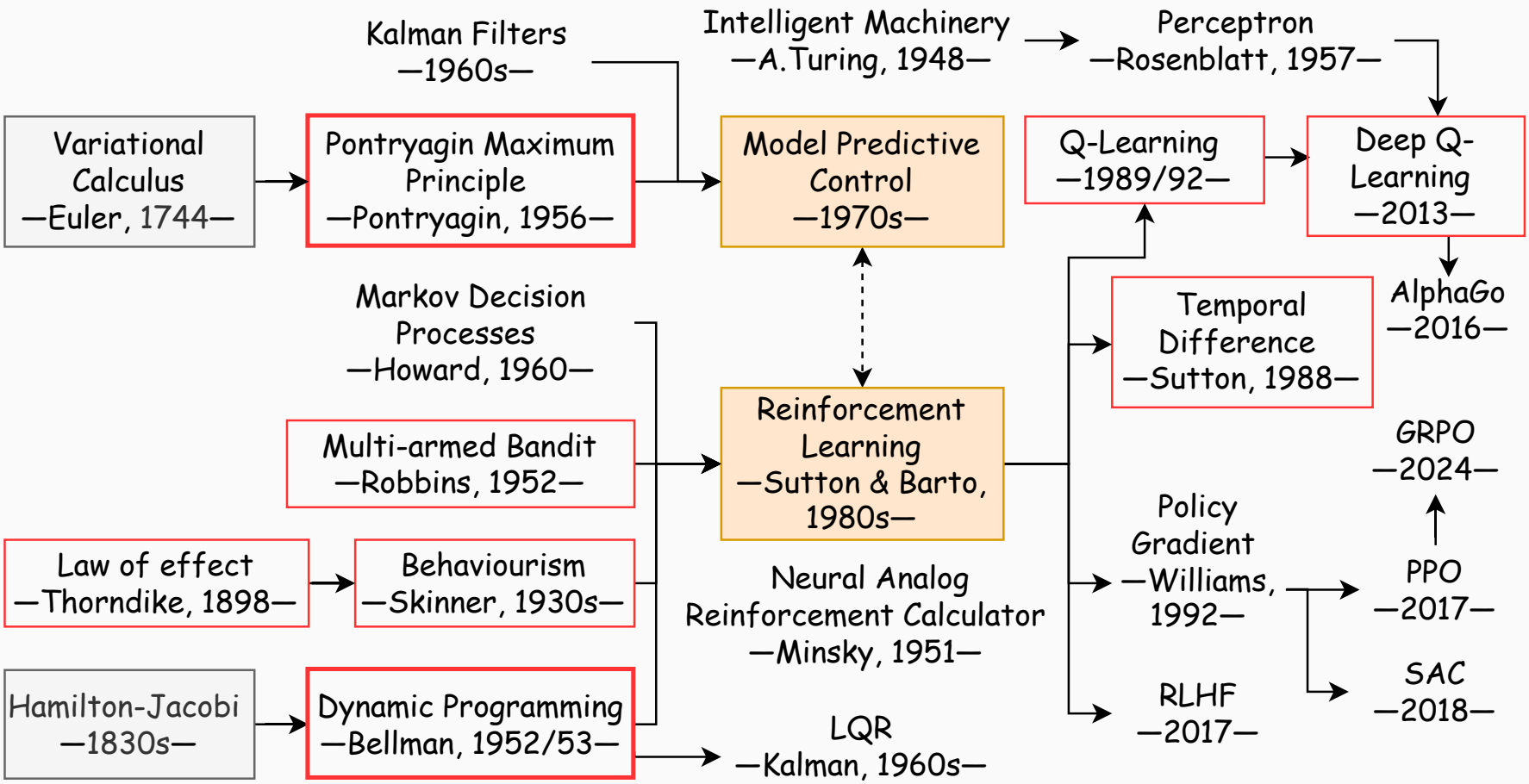


Figure 1: B.F. Skinner

Optimal control and reinforcement learning: timeline



Optimal control vs. reinforcement learning: notation

Concept	Optimal control	Reinforcement learning
State	$x_t \in \mathcal{X}$	$s_t \in \mathcal{S}$
Action	$u_t \in \mathcal{U}$	$a_t \in \mathcal{A}$
Dynamics	$x_{t+1} = f(x_t, u_t)$	$s_{t+1} = f(s_t, a_t)$ [1]
Control/policy	u_t	$a_t \leftarrow \pi(s_t)$
Cost/Reward	$\ell(x_t, u_t)$	$r(s_t, a_t)$
Objective/Value[2]	$J(x_0) := \inf_{u(\cdot)} \sum_{t=0}^{\infty} \gamma^t \ell(x_t, u_t)$	$V(s_0) := \sup_{\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$
Model	f and ℓ are *known*	f and r are *unknown*[3]

[1] Dynamics and policies are usually stochastic.

[2] Can take different forms: finite/infinite horizon, with(out) discount factor γ, etc.

[3] In model-based RL, we know the model/reward or we learn it from data.

Example: Chess (discrete, finite)

	Chess
State space $\mathcal{X}$	Legal board positions, finite with $\lvert\mathcal{X}\rvert \approx 10^{43}$.
Action space $\mathcal{U}(x)$	Legal moves at state x.
Dynamics f	$x_{t+1} = f(x_t, u_t)$, deterministic and known.
Cost/Reward r	$r(x,u) = \begin{cases} +1 & \text{checkmate delivered,} \\ -1 & \text{checkmated,} \\ 0 & \text{otherwise.} \end{cases}$
Objective	$\sup_{u(\cdot)} \sum_{t=0}^{\infty} \gamma^t r(x_t, u_t).$

Example: LQR (continuous, finite)

	Linear Quadratic Regulator
State space $\mathcal{X}$	$x \in \mathbb{R}^n$
Action space $\mathcal{U}$	$u \in \mathbb{R}^m$
Dynamics f	$\dot{x}_t = Ax_t + Bu_t$
Objective[4]	$\inf_{u(\cdot)} \int_0^\infty x^\top Q x + u^\top R u \, \mathrm{d}t, \quad$ with $Q \succeq 0,\ R \succ 0.$

The optimal control problem reduces to an algebraic Riccati equation

$$A^\top P + PA - PBR^{-1}B^\top P + Q = 0,$$

with optimal linear feedback $u^\star = -R^{-1}B^\top P\, x$.

[4] Well-posedness requires a stabilizability and some detectability assumption (see [12]).

Example: Heat equation (continuous, infinite)

	Heat equation with control
State space $\mathcal{X}$	$y(t,\cdot) \in L^2(\Omega)$.
Action space $\mathcal{U}$	$u(t,\cdot)$ supported on $\omega \subset \Omega$.
Dynamics f	$\partial_t y = \Delta y + u\,1_\omega$, with $y\vert_{\partial\Omega} = 0$.
Cost/Reward	$\inf_{u(\cdot)} \int_0^\infty \gamma^t \left(\Vert y - y^\star \Vert_{L^2}^2 + \Vert u \Vert_{L^2}^2 \right) \mathrm{d}t.$

More examples

Problem	**Features**
Poker	Partially observable, stochastic, multi-agent
Portfolio optimization	Continuous, stochastic, constrained
Robotic arm	Continuous, high-dim, partial observation
Inventory control	Discrete, stochastic demand
Autonomous driving	Partial observation, safety-critical

Bellman equation and RL

Optimal Control: A bit of history

In the early 1950s, Richard Bellman was working for the U.S. Air Force on mission planning under uncertainty: optimize the number of aircraft sent on a bombing mission across sequential stages.

The resulting paper "*On the Theory of Dynamic Programming*" [3, 4] laid the foundation for optimal control and reinforcement learning.

Later in the 1950s, Bellman extended this to continuous time, generalizing the *Hamilton–Jacobi equation* from the 1830s–40s.

Figure 2: Richard Bellman

Discrete-time Bellman equation

Dynamics. $x_{t+1} = f(x_t, u_t), \quad x_0 = x.$

Value. $V(x) := \sup_{\pi} \sum_{t=0}^{\infty} \gamma^t r(x_t, u_t).$

Bellman equation. $V(x) = \max_{u \in \mathcal{U}} \{ r(x, u) + \gamma \, V(f(x, u)) \}.$

Optimal policy. $\pi^\star(x) = \arg\max_{u \in \mathcal{U}} \{ r(x, u) + \gamma \, V(f(x, u)) \}.$

Discrete-time Bellman equation

Dynamics. $x_{t+1} = f(x_t, u_t), \quad x_0 = x.$

Value. $V(x) := \sup_{\pi} \sum_{t=0}^{\infty} \gamma^t r(x_t, u_t).$

Bellman equation. $V(x) = \max_{u \in \mathcal{U}} \{ r(x, u) + \gamma V(f(x, u)) \}.$

Proof. Define $\pi^+ := \{u_1, u_2, \ldots\}$ as the tail of π after u_0. Then,

$$
\begin{aligned}
V(x) &:= \sup_{\pi} \sum_{t=0}^{\infty} \gamma^t r(x_t, u_t) \\
&= \max_{u_0 \in \mathcal{U}} \sup_{\pi^+} \left\{ r(x, u_0) + \gamma \sum_{t=0}^{\infty} \gamma^t r(x_{t+1}, u_{t+1}) \right\} \\
&= \max_{u_0 \in \mathcal{U}} \left\{ r(x, u_0) + \gamma \sup_{\pi^+} \sum_{t=0}^{\infty} \gamma^t r(x_{t+1}, u_{t+1}) \right\} \\
&= \max_{u_0 \in \mathcal{U}} \{ r(x, u_0) + \gamma V(f(x, u_0)) \}
\end{aligned}
$$

Discrete-time Bellman equation: Numerical consequences

Dynamics. $x_{t+1} = f(x_t, u_t), \quad x_0 = x.$

Value. $V(x) := \sup_{\pi} \sum_{t=0}^{\infty} \gamma^t r(x_t, u_t).$

Bellman equation. $V(x) = \max_{u \in \mathcal{U}} \{ r(x,u) + \gamma\, V(f(x,u)) \}.$

Theorem (contraction). Let $0 \le \gamma < 1$. The Bellman operator

$$(\mathcal{T}V)(x) := \max_{u \in \mathcal{U}} \{ r(x,u) + \gamma\, V(f(x,u)) \}$$

is a γ-contraction, i.e.,

$$\|\mathcal{T}V - \mathcal{T}W\|_\infty \le \gamma\, \|V - W\|_\infty.$$

Numerical consequence[5].

Converges from any start

$$\|V_k - V^\star\|_\infty \le \gamma^k\, \|V_0 - V^\star\|_\infty,$$
$$\|V_k - V^\star\|_\infty \le \tfrac{\gamma}{1-\gamma}\, \|V_k - V_{k-1}\|_\infty,$$

We stop when $\|V_{k+1} - V_k\|_\infty < \varepsilon$

Proof. Fix x. Using $|\max_u F_u - \max_u G_u| \le \max_u |F_u - G_u|$,

$$|(\mathcal{T}V)(x) - (\mathcal{T}W)(x)| \le \max_{u \in \mathcal{U}} \left|\gamma V(f(x,u)) - \gamma W(f(x,u))\right| \le \gamma\, \|V - W\|_\infty.$$

[5] The cost per iteration is $\mathcal{O}(|\mathcal{X}||\mathcal{U}|)$ and assumes access to r and f.

Continuous-time Bellman (HJB) equation

	Discrete time[6]	Continuous time[7]
Dynamics	$x_{t+1} = f(x_t, u_t)$	$\dot{x}_t = f(x_t, u_t)$
Value	$V(x) = \sup_{u_t} \sum_{t=0}^{\infty} \gamma^t \, r(x_t, u_t)$	$V(x) = \sup_{u(\cdot)} \int_0^{\infty} \gamma^t \, r(x_t, u_t) \, dt$
Bellman	$V(x) = \max_u \left\{ r(x,u) + \gamma \, V(f(x,u)) \right\}$	$-\ln(\gamma) \, V(x) = \max_u \left\{ r(x,u) + \nabla V(x)^\top f(x,u) \right\}$

Intuition. As $\Delta t \to 0$, by the discrete-time Bellman equation

$$\begin{aligned} V(x) &= \max_u \left\{ r(x,u)\Delta t + \gamma^{\Delta t} V\big(x + f(x,u)\Delta t\big) \right\} \\ &= \max_u \left\{ r(x,u)\Delta t + (1 + \ln(\gamma)\Delta t) \left[V(x) + \nabla V(x)^\top f(x,u)\Delta t \right] \right\}. \end{aligned}$$

Cancel $V(x)$, divide by Δt, and let $\Delta t \to 0$:

$$0 = \max_{u \in \mathcal{U}} \left\{ r(x,u) + \nabla V(x)^\top f(x,u) + \ln(\gamma) V(x) \right\}$$

The usual convention in continuous-time is to write the discount factor as $\gamma \equiv e^{-\rho}$.

[6] Most research focuses on the discrete and stochastic case.

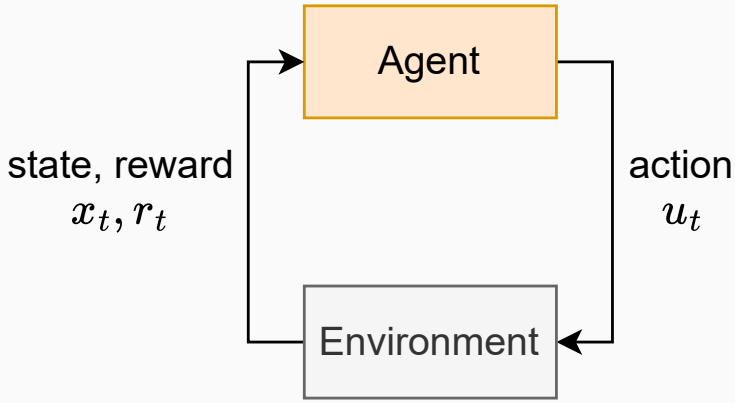
[7] Recent work (2021-) on continuous-time RL [23, 11, 10] shows great potential.

Reinforcement Learning

From control to RL

In reinforcement learning, the agent

- observes states x_t,
- chooses actions $u_t \sim \pi(\cdot \mid x_t)$,
- receives rewards $r_t = r(x_t, u_t)$,
- observes next states $x_{t+1} = f(x_t, u_t)$.



Hence, we have $\{x_t, u_t, r_t, x_{t+1}\}_t$ data but no access to f or r.

Two families of algorithms:

- Value-based: approximate V, recover π greedily.
- Policy-based: parametrize π_θ directly.

Q-learning: Bellman

Bellman equation

$$V(x) = \max_u \Big\{ r(x,u) + \gamma\, V(f(x,u)) \Big\}.$$

Let us define the Q-function

$$Q(x,u) := r(x,u) + \gamma\, V(f(x,u)).$$

Then $V(x) = \max_{u \in \mathcal{U}} Q(x,u)$, and hence:

$$\boxed{Q(x,u) = r(x,u) + \gamma \max_{u'} Q(f(x,u),u').}$$

The *greedy policy* is optimal

$$\pi^\star(x) \in \arg\max_u Q^\star(x,u)$$

Define the Bellman operator for Q:

$$(\mathcal{T}Q)(x,u) := r(x,u) + \gamma \max_{u' \in \mathcal{U}} Q(f(x,u),u').$$

Theorem (contraction). For all Q_1, Q_2, $\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty \le \gamma \, \|Q_1 - Q_2\|_\infty$.

Proof. Using $|\max_{u'} F_{u'} - \max_{u'} G_{u'}| \le \max_{u'} |F_{u'} - G_{u'}|$,

$$\begin{aligned} &|(\mathcal{T}Q_1)(x,u) - (\mathcal{T}Q_2)(x,u)| \\ &\le \gamma \max_{u'} |Q_1(f(x,u),u') - Q_2(f(x,u),u')| \\ &\le \gamma \, \|Q_1 - Q_2\|_\infty. \end{aligned}$$

Numerical consequence[8].

Converges from any start

$$\|Q_k - Q^\star\|_\infty \le \gamma^k \, \|Q_0 - Q^\star\|_\infty,$$

We stop when $\|Q_{k+1} - Q_k\|_\infty < \varepsilon$.

[8] The cost per iteration is $\mathcal{O}(|\mathcal{S}||\mathcal{U}|^2)$ and assumes access to f and r.

Q-learning: Bellman

Bellman equation

$$V(x) = \max_{u \in \mathcal{U}} \Big\{ r(x,u) + \gamma\, V(f(x,u)) \Big\}.$$

Let us define the Q-function

$$Q(x,u) := r(x,u) + \gamma\, V(f(x,u)).$$

Then $V(x) = \max_{u \in \mathcal{U}} Q(x,u)$, and hence:

$$Q(x,u) = r(x,u) + \gamma \max_{u'} Q(f(x,u), u').$$

The optimal policy is

$$\pi^\star(x) = \arg\max_{u \in \mathcal{U}} Q(x,u).$$

```
import torch
import torch.nn as nn

Q = nn.Linear(d_state, n_actions)
opt = torch.optim.Adam(Q.parameters())

for x, a, r, x_next in buffer:
    q_sa = Q(x)[a]
    with torch.no_grad():
        q_next = Q(x_next).max()
        target = r + gamma * q_next
    loss = (q_sa - target).pow(2)
    opt.zero_grad()
    loss.backward()
    opt.step()
```

Initialize Q (e.g., neural net). The update rule[9]: $Q \leftarrow Q + \alpha\delta_t$ where:

$$\delta_t \equiv r_t + \gamma \max_{u'} Q(x_{t+1}, u') - Q(x_t, u_t). \tag{1}$$

[9] Converges to Q-function if every state–action pair occurs infinitely often [20].

Q-learning: Bellman

Bellman equation

$$V(x) = \max_{u \in \mathcal{U}} \Big\{ r(x,u) + \gamma\, V(f(x,u)) \Big\}.$$

Let us define the Q-function

$$Q(x,u) := r(x,u) + \gamma\, V(f(x,u)).$$

Then $V(x) = \max_{u \in \mathcal{U}} Q(x,u)$, and hence:

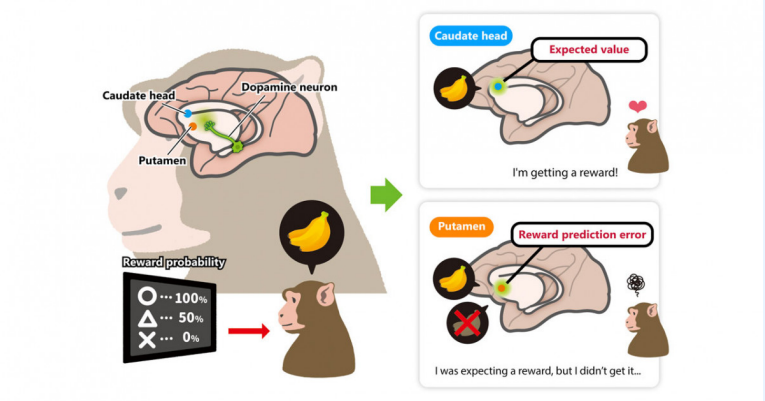
$$Q(x,u) = r(x,u) + \gamma \max_{u'} Q(f(x,u), u').$$

The optimal policy is

$$\pi^\star(x) = \arg\max_{u \in \mathcal{U}} Q(x,u).$$

Link to dopamine

In the 1990s [17], dopamine neurons were shown to encode a *reward-prediction error* as in (2): firing strongly for better-than-expected rewards and dipping when expected rewards fail to appear.



Initialize Q. Use the update rule[10]: $Q \leftarrow Q + \alpha\delta_t$ where:

$$\delta_t \equiv r_t + \gamma \max_{u'} Q(x_{t+1}, u') - Q(x_t, u_t). \tag{2}$$

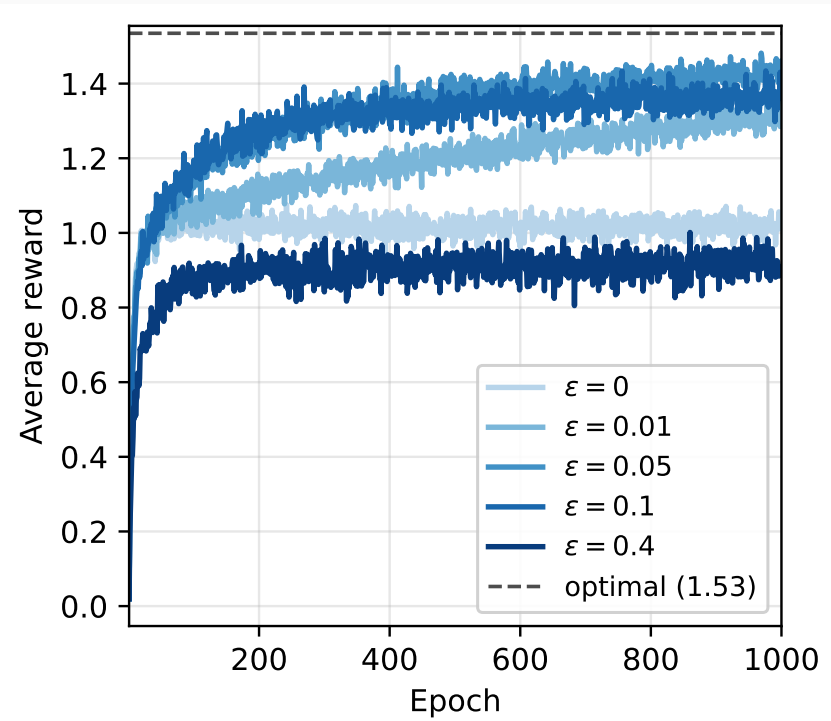
[10] Converges to Q-function if every state–action pair occurs infinitely often [20]

Exploration vs Exploitation

Action-sampling rule used to collect data:

$$u_t \sim \begin{cases} \arg\max_{u'} Q(x_t, u'), & \text{with probability } 1-\varepsilon, \\ \text{Uniform}(\mathcal{U}), & \text{with probability } \varepsilon. \end{cases}$$

- $\varepsilon \to 0$: pure exploitation; fast but might not find the optimal solution.
- $\varepsilon \to 1$: pure exploration; unbiased but slow.



Example (Multi-armed bandit [15]): 1 state, reward $r(u) \sim \mathcal{N}(\mu_u, 1)$ for $u = 1, \ldots, |\mathcal{U}|$.

$$V^\star(x) = \max_u \mu_u.$$

Q-learning estimates $Q(u) \to \mu_u$ from sampled rewards.

On-policy vs. off-policy

The behavior policy μ collects the data; the target policy π is the one we train.

- On-policy ($\mu = \pi$): learn from *your own* current mistakes.
- Off-policy ($\mu \neq \pi$): learn from *anyone's* mistakes (replay buffer).

	On-policy	**Off-policy**
Examples	SARSA, REINFORCE, A2C/A3C, TRPO, PPO	Q-learning, DQN, DDPG, TD3, SAC
Data reuse	Discard after each update	Replay buffer; reuse old data
Sample efficiency	Low	High
Stability	More stable, simpler theory	Can diverge[11]; *deadly triad* [18].

[11] Needs tricks (target nets, importance sampling)

Pontryagin and MPC

Optimal Control: A bit of history

Lev Pontryagin was a Soviet mathematician who led the group working on trajectory optimization for *ballistic missiles* and rocket flight in the 1950s, during the Cold War.

Out of this work came the Maximum Principle [13], giving necessary conditions for optimality via the adjoint (costate) equation.

The same theory explains *bang-bang* controls; Pontryagin is also known for Pontryagin duality (Fourier analysis).

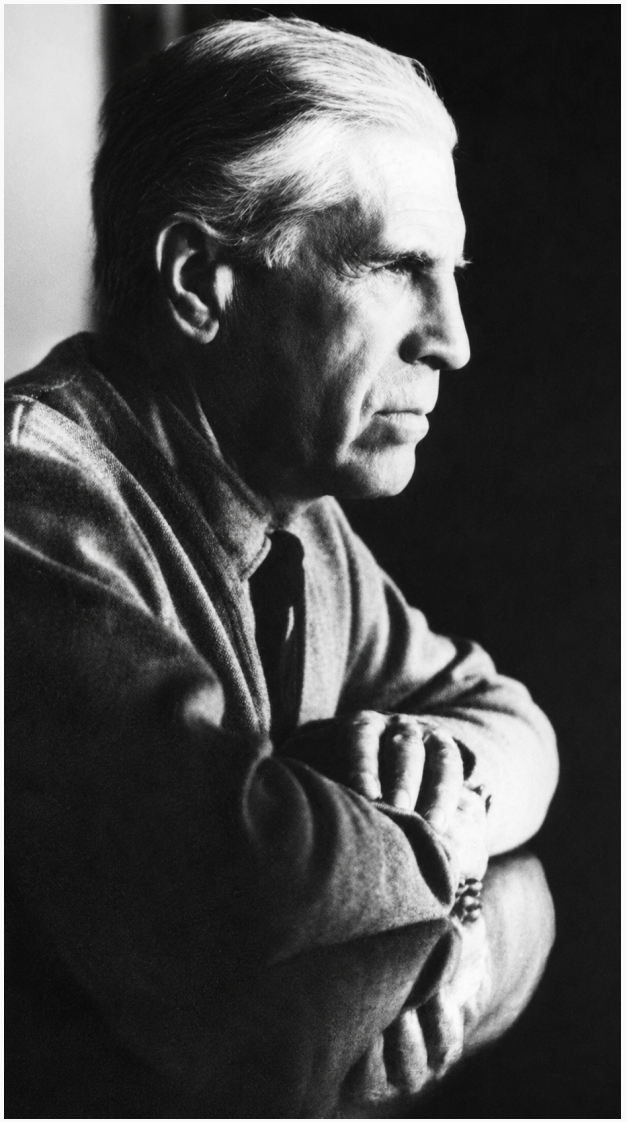


Figure 3: Lev Pontryagin

Pontryagin's Maximum Principle (PMP)

Finite-horizon problem.

$$\min_{u(\cdot)} \; J(u) = \int_0^T \ell(x_t, u_t)\, dt + \ell_T(x_T), \qquad \dot{x}_t = f(x_t, u_t), \;\; x_0 = x.$$

Hamiltonian. $H(x, u, p) := \ell(x, u) + p^\top f(x, u)$.

Necessary conditions [13]. An optimal pair $(x^\star, u^\star)$ satisfies,

$$\dot{x}_t^\star = \nabla_p H \;=\; f(x_t^\star, u_t^\star), \qquad x_0^\star = x,$$

$$\dot{p}_t = -\nabla_x H \;=\; -\nabla_x \ell(x_t^\star, u_t^\star) - \big(\nabla_x f(x_t^\star, u_t^\star)\big)^\top p_t, \quad p_T = \nabla \ell_T(x_T^\star),$$

$$u_t^\star = \arg\min_{u \in \mathcal{U}} \; H(x_t^\star, u, p_t).$$

	HJB	**PMP**
Unknown	$V(t,x)$ on $[0,T] \times \mathcal{X}$	$(x_t^\star, p_t)$ on $[0,T]$
Equation	PDE on $\mathcal{X}$	ODE/PDE on $\mathcal{X}^2$
Yields	feedback law $u^\star(x)$ (closed-loop)	control signal $u^\star(t)$ (open-loop)
Conditions	Necessary and sufficient	Necessary only

MPC: Shooting method via the adjoint

Goal: Find u that minimizes $J(u_t)$ using gradient-based optimization.

	Continuous adjoint	**Discrete adjoint**[12]
State	$\dot{x}_t = f(x_t, u_t)$	$x_{t+1} = x_t + \Delta t\, f(x_t, u_t)$
Adjoint	$\dot{p}_t = -\nabla_x \ell_t - (\nabla_x f_t)^\top p_t,$	$\hat{p}_t = \hat{p}_{t+1} + \Delta t\, [\nabla_x \ell_t + (\nabla_x f_t)^\top \hat{p}_{t+1}],$
	$p_T = \nabla \ell_T(x_T)$	$\hat{p}_N = \nabla \ell_T(x_N)$
Gradient	$\nabla_{u(t)} J = \nabla_u \ell_t + (\nabla_u f_t)^\top p_t$	$\nabla_{u_t} J = \Delta t\, [\nabla_u \ell_t + (\nabla_u f_t)^\top \hat{p}_{t+1}]$

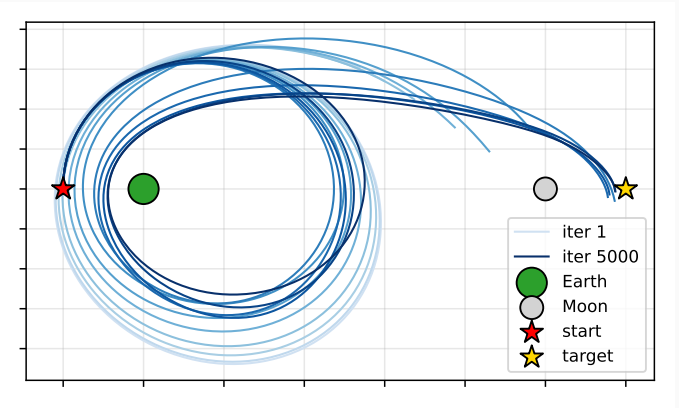
Shooting method.

- Direct shooting. (1) Forward simulate $x_0, \ldots, x_N$ from u. (2) Backward sweep $\hat{p}_N, \ldots, \hat{p}_0$. (3) Update $u_t \leftarrow u_t - \eta \nabla_{u_t} J$. Repeat.
- Indirect shooting. Guess p_0, integrate $(\dot{x}, \dot{p})$ with $u^\star = \arg\min_u H$, Newton-update p_0 until $p_T = \nabla \ell_T(x_T)$.

[12] Most popular in practice because it is easier to implement and numerically stable.

Example: Earth-to-Moon transfer trajectory

State	Position q, velocity v ($x \in \mathbb{R}^4$).
Control	Thrust acceleration ($u \in \mathbb{R}^2$).
Dynamics	$\dot{q} = v,$
	$\dot{v} = -\mu_E \dfrac{q - q_E}{\|q - q_E\|^3} - \mu_M \dfrac{q - q_M}{\|q - q_M\|^3} + u.$
Running cost	$\ell(x, u) = \frac{1}{2}\|u\|^2$ (fuel).
Terminal cost	$\ell_T(x_T) = \frac{\alpha}{2}\|q_T - q^\star\|^2 + \frac{\beta}{2}\|v_T - v^\star\|^2.$




```
mu_E, mu_M = 1.0, 0.012
qE, qM=torch.tensor([[0., 0.],[1., 0.]])

def f(x, u):
    q, v = x[:2], x[2:]
    dE, dM = q - qE, q - qM
    aE = -mu_E * dE / dE.norm()**3
    aM = -mu_M * dM / dM.norm()**3
    return torch.cat([v, aE + aM + u])

x0 = torch.tensor([0.9, 0.1, 0.0, 1.05])
N, dt = 400, 0.02
u = torch.zeros(N, 2, requires_grad=True)
opt = torch.optim.Adam([u], lr=5e-3)
```

```
a, b = 50.0, 10.0

for step in range(5000):
  x, J = x0, 0.0
  for t in range(N):
      J += 0.5 * dt * (u[t]**2).sum()
      x += dt * f(x, u[t]) # Euler
  # terminal: return to start
  J += 0.5 * a * ((x[:2] - x0[:2])**2).sum()
  J += 0.5 * b * ((x[2:] - x0[2:])**2).sum()

  opt.zero_grad()
  J.backward() # discrete adjoint
  opt.step()
```

Extra: MPC $\approx$ Neural ODE training

	MPC	**Neural ODE** [5]
Dynamics	$\dot{x} = f(x, u)$ where f *known*	$\dot{x} = f(x, \theta)$ where $f(\cdot, \theta)$ neural net
Optimize	control $u(\cdot)$	parameters θ
Loss	$\int_0^T \ell(x, u)\, dt + \ell_T(x_T)$	data fit $\sum_i \|\hat{x}_{t_i} - x_i^{\text{obs}}\|^2$
Adjoint	$\nabla_{u(t)} J = \nabla_u H$	$\nabla_\theta J = \int_0^T (\nabla_\theta f)^\top p_t\, dt$

Extra: non-autonomous, stochastic, finite-horizon

Non-autonomous. $\dot{x}_t = f(t, x_t, u_t)$.

$$-\ln(\gamma)V - \partial_t V = \max_{u \in \mathcal{U}} \left\{ r(t,x,u) + \nabla_x V(t,x)^\top f(t,x,u) \right\}.$$

Finite-horizon. Terminal payoff $g(x)$. Then, $V(T,x) = g(x)$ and

$$-\ln(\gamma)V - \partial_t V = \max_{u \in \mathcal{U}} \left\{ r(t,x,u) + \nabla_x V(t,x)^\top f(t,x,u) \right\}.$$

Stochastic. $dX_t = f(X_t, u_t)\,dt + \sigma(X_t, u_t)\,dW_t$.

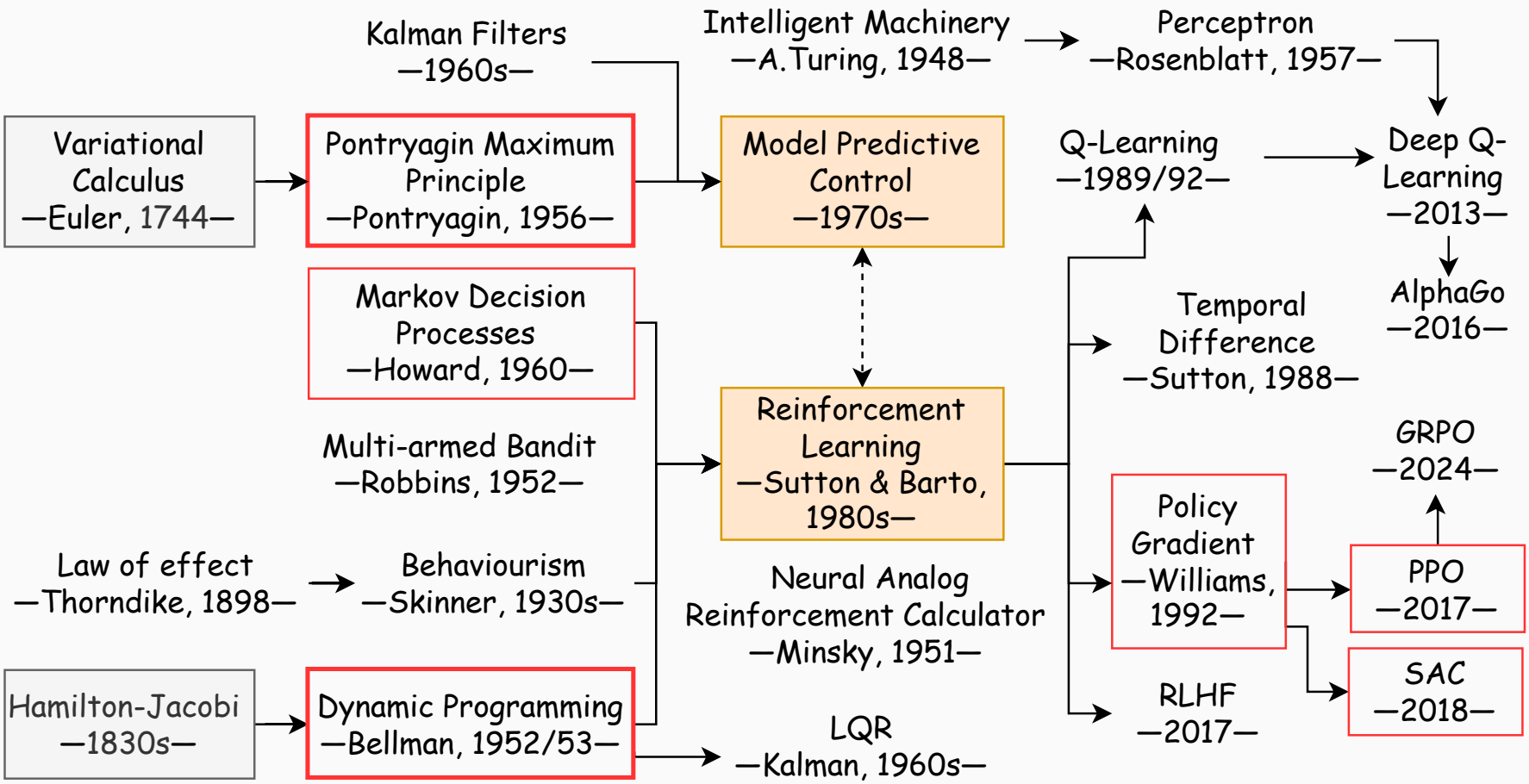
$$-\ln(\gamma)V(x) = \max_{u \in \mathcal{U}} \left\{ r(x,u) + \nabla V(x)^\top f(x,u) + \tfrac{1}{2}\mathrm{tr}\left(\sigma\sigma^\top D^2 V(x)\right) \right\}.$$

Infinite-dimensional [6]. $\dot{x}_t = Ax_t + b(x_t, u_t)$ where A generates a C_0-semigroup on a Hilbert space H and $b : H \times \mathcal{U} \to H$.

$$-\ln(\gamma)V(x) = \sup_{u \in \mathcal{U}} \{ r(x,u) + \langle Ax + b(x,u),\, DV(x) \rangle_H \}.$$

Day 2: Modern Reinforcement Learning

Optimal control and reinforcement learning: timeline



Markov Decision Process

A Markov Decision Process (MDP) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$:

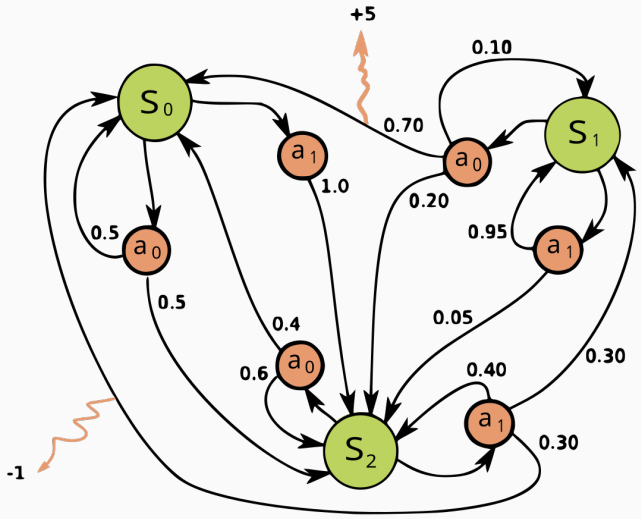
- **State space** $(\mathcal{S}, \mathcal{F}_{\mathcal{S}})$: a measurable space.
- **Action space** $(\mathcal{A}, \mathcal{F}_{\mathcal{A}})$: a measurable space.
- **Transition probability** $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$, $P(\cdot \mid s, a)$ is a probability measure on $\mathcal{S}$, and $(s, a) \mapsto P(B \mid s, a)$ is measurable $\forall B \in \mathcal{F}_{\mathcal{S}}$:

$$P(B \mid s, a) = \mathbb{P}\big(S_{t+1} \in B \,\big|\, S_t = s,\, A_t = a\big).$$

- **Reward** $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, measurable. It is the expected one-step reward

$$r(s, a) = \mathbb{E}\big[R_{t+1} \,\big|\, S_t = s,\, A_t = a\big].$$

- **Discount factor** $\gamma \in [0, 1)$ used to weight future rewards.



Markov property

For every measurable $B \subseteq \mathcal{S}$,

$$\mathbb{P}(B \mid S_0, A_0, \ldots, S_t, A_t) = P(B \mid S_t, A_t).$$

The future depends on the past only through the present state and action.

Markov Decision Process: Policy, Value, Bellman

Policy. A (stationary, Markov) *policy* is

$$\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A}),$$

where for each $s \in \mathcal{S}$, $\pi(\cdot \mid s)$ is a probability measure on $\mathcal{A}$, and $s \mapsto \pi(B \mid s)$ is measurable for every $B \in \mathcal{F}_{\mathcal{A}}$.

Value functions. For a policy π,

$$V^{\pi}(s) \;:=\; \mathbb{E}_s^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \, R_{t+1}\right], \quad Q^{\pi}(s,a) \;:=\; \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \, R_{t+1} \,\middle|\, S_0 = s,\, A_0 = a\right].$$

The optimal value and action-value functions are

$$V^{\star}(s) \;=\; \sup_{\pi} V^{\pi}(s), \qquad Q^{\star}(s,a) \;=\; \sup_{\pi} Q^{\pi}(s,a).$$

Bellman.

$$V^{\star}(s) \;=\; \sup_{a \in \mathcal{A}} \left\{ r(s,a) \;+\; \gamma \int_{\mathcal{S}} V^{\star}(s') \, P(ds' \mid s,a) \right\}.$$

Notation: Day 1 (deterministic) vs. Day 2 (probabilistic)

	Day 1: deterministic	Day 2: probabilistic
State	$s_t \in \mathcal{S}$	$s_t \in \mathcal{S}$
Action	$a_t \in \mathcal{A}$	$a_t \in \mathcal{A}$
Dynamics	$s_{t+1} = f(s_t, a_t)$	$S_{t+1} \sim P(\cdot \mid S_t, A_t)$
Policy	$a_t \leftarrow \pi(s_t)$	$A_t \sim \pi(\cdot \mid S_t)$
Reward	$r(s_t, a_t)$	$r(s_t, a_t) = \mathbb{E}[R_{t+1} \mid S_t = s_t, A_t = a_t]$
Value	$V^\pi(s) = \sum_{t \geq 0} \gamma^t r(s_t, a_t)$	$V^\pi(s) = \mathbb{E}_s^\pi \left[\sum_{t \geq 0} \gamma^t R_{t+1} \right]$
Bellman	$V^*(s) = \sup_a \{ r(s,a) + \gamma V^*(f(s,a)) \}$	$V^\star(s) = \sup_a \left\{ r(s,a) + \gamma \int_{\mathcal{S}} V^\star(s') P(ds' \mid s, a) \right\}$

Policy gradient theorem

Setup. Policy $\pi_\theta(a \mid s)$, initial law ν, transition kernel $P(\cdot \mid s, a)$, discount $\gamma \in [0, 1)$. The trajectory $\tau = (S_0, A_0, S_1, A_1, \dots)$ has density

$$p_\theta(\tau) \;=\; \nu(S_0) \prod_{t \geq 0} \pi_\theta(A_t \mid S_t)\, P(S_{t+1} \mid S_t, A_t).$$

Write the trajectory return and objective

$$G(\tau) := \sum_{k \geq 0} \gamma^k R_{k+1}, \qquad J(\theta) := \mathbb{E}_{\tau \sim p_\theta}[G(\tau)],$$

Theorem (Policy gradient [19]).

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t \geq 0} \gamma^t \, \nabla_\theta \log \pi_\theta(A_t \mid S_t)\; Q_{\pi_\theta}(S_t, A_t) \right].$$

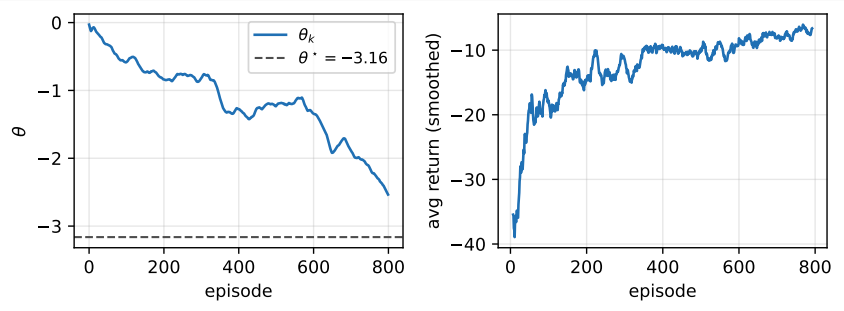
Recall $Q_{\pi_\theta}(s, a) := \mathbb{E}^{\pi_\theta}\left[\sum_{k \geq 0} \gamma^k R_{k+1} \,\middle|\, S_0 = s,\, A_0 = a \right]$.

REINFORCE algorithm [21] — on-policy

1. Roll out $\{s_t, a_t, r_{t+1}\}_{t=0}^{N}$ with $a_t \sim \pi_\theta(\cdot \mid s_t)$.
2. Compute returns (Monte-Carlo) $G_t = \sum_{k \geq 0} \gamma^k \, r_{t+k+1}$
3. Update

$$\theta \leftarrow \theta + \eta \sum_t \gamma^t \, \nabla_\theta \log \pi_\theta(a_t \mid s_t) \, G_t.$$

Dynamics	$\dot{x} = u, \quad x \in \mathbb{R}$
Cost	$\ell(x, u) = x^2 + 0.1\, u^2$
Policy	$\pi_\theta(u \mid x) = \mathcal{N}(\theta\, x,\ \sigma^2)$
LQR opt.	$\theta^\star = -\sqrt{q/r} = -\sqrt{10}$




```python
import torch, torch.nn as nn
from torch.distributions import Normal

policy = nn.Sequential(
    nn.Linear(d_state, 32), nn.Tanh(),
    nn.Linear(32, d_action))
opt = torch.optim.Adam(policy.parameters())

for ep in range(N_EP):
    log_p, rew = [], []
    x = env.reset()
    for t in range(T):
        dist = Normal(policy(x), sigma)
        u = dist.sample()
        log_p.append(dist.log_prob(u).sum())
        x, r = env.step(u)
        rew.append(r)
    G = discounted_returns(rew, gamma)
    loss = -(torch.stack(log_p) * G).sum()
    opt.zero_grad()
    loss.backward()
    opt.step()
```

Proximal Policy Optimization (PPO) [16] — on-policy[13]

PPO trains a policy π_θ (actor) together with a value network V_ϕ (critic).

Advantage. From the residual $\delta_t = r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$, the advantage

$$A_t := \sum_{l \geq 0} (\gamma\lambda)^l \, \delta_{t+l}$$

estimates how much better a_t is than average. If $A_t > 0$, the action is better than average; if $A_t < 0$, it is worse.

Value loss. Fits V_ϕ to the return $R_t = A_t + V_\phi(s_t)$:

$$L^{\mathrm{VF}} := \mathbb{E}_t[(V_\phi(s_t) - R_t)^2].$$

```
import torch, torch.nn as nn

policy = nn.Sequential(
       nn.Linear(d_state, 32), nn.Tanh(),
       nn.Linear(32, d_action))
value = nn.Sequential(
       nn.Linear(d_state, 32), nn.Tanh(),
       nn.Linear(32, 1))
sigma = 0.5 # action std
opt = torch.optim.Adam([*policy.parameters()
     , *value.parameters()], lr=3e-4)
eps, c1, c2 = 0.2, 0.5, 0.01 # clip, VF, ent
gamma, lam, K_EPOCHS = 0.99, 0.95, 10
```

Entropy. Rewards exploration

$$\mathcal{H}[\pi_\theta] := -\mathbb{E}_{a \sim \pi_\theta}[\log \pi_\theta(a \mid s_t)]$$

[13] On-policy, but each iteration's batch is reused for K epochs.

Proximal Policy Optimization (PPO) [16] — on-policy

Clipped surrogate. Repeat for K epochs:

1. Roll out with $\pi_{\theta_{\text{old}}}$; estimate advantages A_t.
2. Probability ratio
$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$
3. Clipped surrogate objective

$$L^{\text{CLIP}}(\theta) := \mathbb{E}_t\big[\min\left(r_t A_t,\ \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\, A_t\right)\big]$$

The clip removes the incentive to move r_t beyond $[1-\epsilon,\ 1+\epsilon]$.

Full loss

$$\mathcal{L} := -L^{\text{CLIP}} + c_1 L^{\text{VF}} - c_2 \mathcal{H}[\pi_\theta]$$

```
import torch
from torch.distributions import Normal

# rollout under pi_old -> states, actions,
# old_log_p, advantages A, returns R
for _ in range(K_EPOCHS):
  dist = Normal(policy(states), sigma)
  log_p = dist.log_prob(actions).sum(-1)
  ratio = torch.exp(log_p - old_log_p)

  s1 = ratio * A
  s2 = torch.clamp(ratio, 1-eps, 1+eps) * A
  pi_loss = torch.min(s1, s2).mean()

  v_loss = ((value(states) - R)**2).mean()
  ent = dist.entropy().sum(-1).mean()
  loss = -pi_loss + c1 * v_loss - c2 * ent

  opt.zero_grad()
  loss.backward()
  opt.step()
```

Experiments: Max Golightly

Soft Policy Iteration

- Add an entropy term to the value function:
  $V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \log \pi(a_t|s_t)]$
- Bellman operator: $\mathcal{T}^\pi Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V(s_{t+1})]$
- Convergence result [9]: $\pi^k \to \pi^*$

Soft Actor-Critic

Soft Actor-Critic consists of three parameterized networks approximating the value, Q, and policy functions.

- Define the induced Q-function:
  $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim p} \left[V_{\bar{\psi}}(s_{t+1}) \right]$

Objective functions:

- Train the parameterized Q_θ against $\hat{Q}$:
  $J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t) \right)^2 \right]$
- Train the parameterized V_ψ against V as defined before:
  $J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t | s_t)])^2]$
- Train the policy to maximize Q:
  $J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D},\, \epsilon_t \sim \mathcal{N}}[\log \pi_\phi(f_\phi(\epsilon_t; s_t) \mid s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))] \, .$

Soft Actor-Critic

Monte Carlo gradient estimators:

- $\widehat{\nabla_\psi J_V}(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\theta(s_t, a_t) + \log \pi_\phi(a_t|s_t))$
- $\widehat{\nabla_\theta J_Q}(\theta) = \nabla_\theta Q_\theta(s_t, a_t) \left(Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_{\bar{\psi}}(s_{t+1})\right).$
- $\widehat{\nabla_\phi J_\pi(\phi)} = \nabla_\phi \log \pi_\phi(a_t \mid s_t) + \left(\nabla_{a_t} \log \pi_\phi(a_t \mid s_t) - \nabla_{a_t} Q_\theta(s_t, a_t)\right) \nabla_\phi f_\phi(\epsilon_t; s_t)$

Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a_t \mid s_t)$
    $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \widehat{\nabla_\psi J_V(\psi)}$
    $\theta_i \leftarrow \theta_i - \lambda_Q \widehat{\nabla_{\theta_i} J_Q(\theta_i)}$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \widehat{\nabla_\phi J_\pi(\phi)}$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

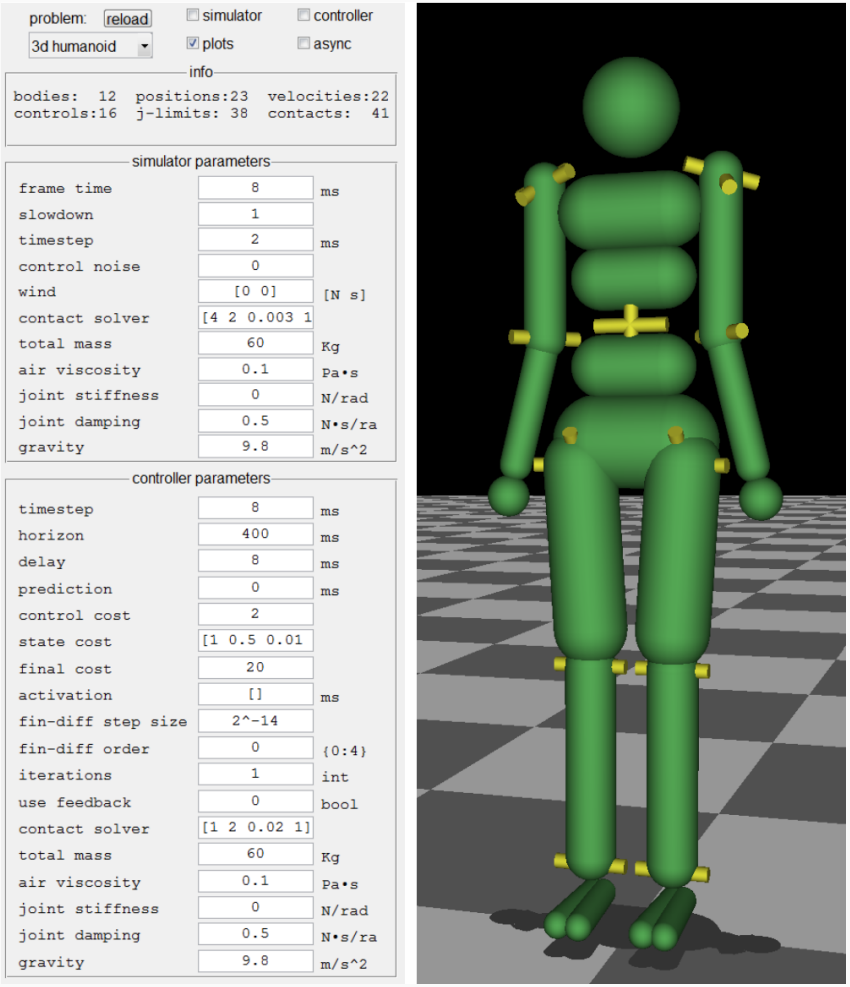
Simulation Example: State Space

```
Box(-inf, inf, (376,), float64)
```

So the state space is $\mathcal{S} = \mathbb{R}^{376}$.

Vector Components:

- **Kinematics (Movement):**
  - Joint positions (angles)
  - Joint velocities (angular velocity)
  - Center of Mass (CoM) velocity
- **Dynamics (Physics):**
  - Actuator forces (motor output)
  - External contact forces (e.g. ground contact)
  - Mass & Inertia relative to CoM



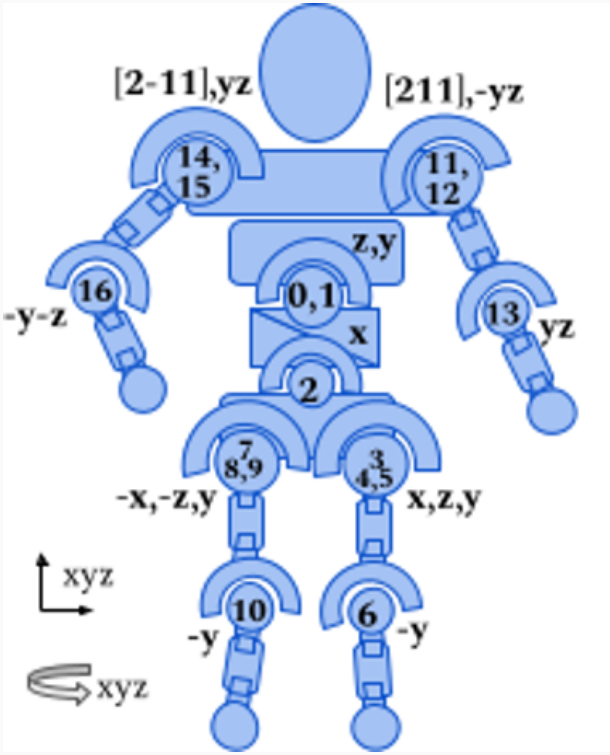
Source: [7]

Simulation Example: Action Space

Definition:

```
Box(-0.4, 0.4, (17,), float32)
```

- **Continuous Control:** The agent outputs a 17-dimensional vector of values in $[-0.4, 0.4]$, one per degree of freedom. So $\mathcal{A} = [-0.4, 0.4]^{17}$.
- **Physical Interpretation:** These values act as normalized torques applied to the hinge joints $\rightarrow$ later mapped to physical torques by the simulator



Source: [7]

System Dynamics

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = Bu + J^T(q)F_{ext}$$

1. Physical Terms

- $M(q)$: Inertia Matrix
  (Mass distribution dependent on pose)
- $C(q, \dot{q})$: Coriolis & Centrifugal forces
  (Effects of rotation velocity)
- $G(q)$: Gravitational Vector
  (Forces pulling down)
- $J^T(q)$: Transpose Jacobian
  (Projects ext. forces F_{ext} to joints)

2. RL Interface

- **Observation Space**
  - q: Joint Positions (Angles)
  - $\dot{q}$: Joint Velocities
- **Action Space**
  - u: Desired Motor Torques
  - B: Actuation Matrix

Source: [22] *The physics engine computes $\ddot{q}$ based on these terms to advance the simulation.*

Reward Function: The Objective

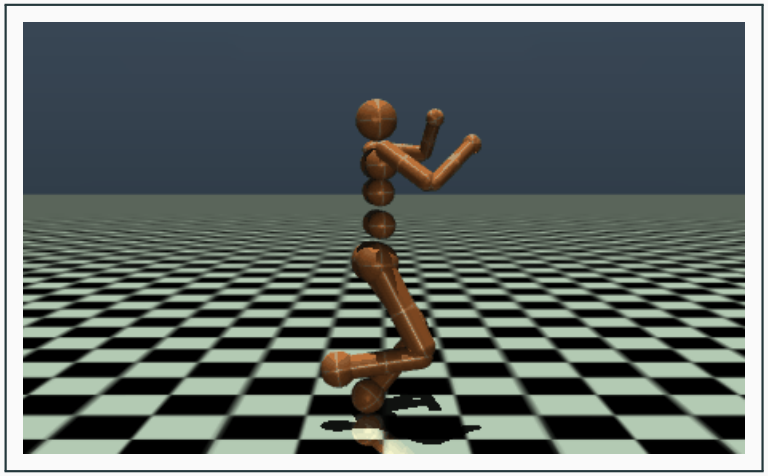
$$R = \underbrace{\textit{healthy} + \textit{forward}}_{\textbf{Incentives}} - \underbrace{\textit{control} - \textit{impact}}_{\textbf{Penalties}}$$

Component	Description	Formula
Healthy	Bonus for every timestep the robot does not fall (survival).	$+w_{hlth}$
Forward	Reward for moving forward (velocity along x-axis).	$+w_{fwd} \cdot \frac{x_{t+\mathbf{1}} - x_t}{\Delta t}$
Control Cost	Penalty for large motor forces (energy efficiency).	$-w_{ctrl} \cdot \|\mathrm{u}_t\|_2^2$
Contact Cost	Penalty for heavy impacts with the ground to prevent damage.	$-w_{imp} \cdot \|\mathrm{F}_{ext}\|_2^2$

$$r_t = w_{hlth} + w_{fwd} \cdot \frac{x_{t+1} - x_t}{\Delta t} - \left(w_{ctrl}\|\mathrm{u}_t\|_2^2 + w_{imp}\|\mathrm{F}_{ext}\|_2^2\right)$$

Source: [7]

Termination Conditions



Scenario: torso height drops below the threshold

1. Unhealthy State (Terminated)

The episode ends immediately if the robot is no longer upright.

- **Condition:** Torso z-coordinate (height) leaves the safe range.
- **Result:** Accumulation of rewards stops.

2. Time Limit (Truncated)

The simulation also stops if the maximum duration is reached.

- Prevents infinitely long episodes when a policy is successful.

Source: [7]

Implementation

- Value network: 3 fully connected layers with ReLU activation
- Q network: 2 networks of 3 fully connected layers with ReLU activation; picks the lower of the two
- Policy network: the mean network consists of 3 fully connected layers with ReLU activation, and the standard-deviation network is a single fully connected layer. Idea: State-Dependent Exploration [14]

Implementation

train.py > ...

```python
import gymnasium as gym
from stable_baselines3 import SAC

env = gym.make("Humanoid-v5", render_mode="human")

agent = SAC("MlpPolicy", env, verbose=1)

agent.learn(total_timesteps=1_000)
agent.save("sac_walker2d")

n_steps = 1000

observation, info = env.reset()

total_reward = 0

print("Action space: ",env.action_space)
for _ in range(n_steps):

    action, _ = agent.predict(observation, deterministic=True) #sample next action

    # Take the action and see what happens
    observation, reward, terminated, truncated, info = env.step(action)
    env.render()
    total_reward += reward
    if terminated or truncated:
        observation, info = env.reset()

print(f"Episode finished! Total reward: {total_reward}")

env.close()
```

Results

- Two videos of the robot
- Live training data example

Outlook and Discussion

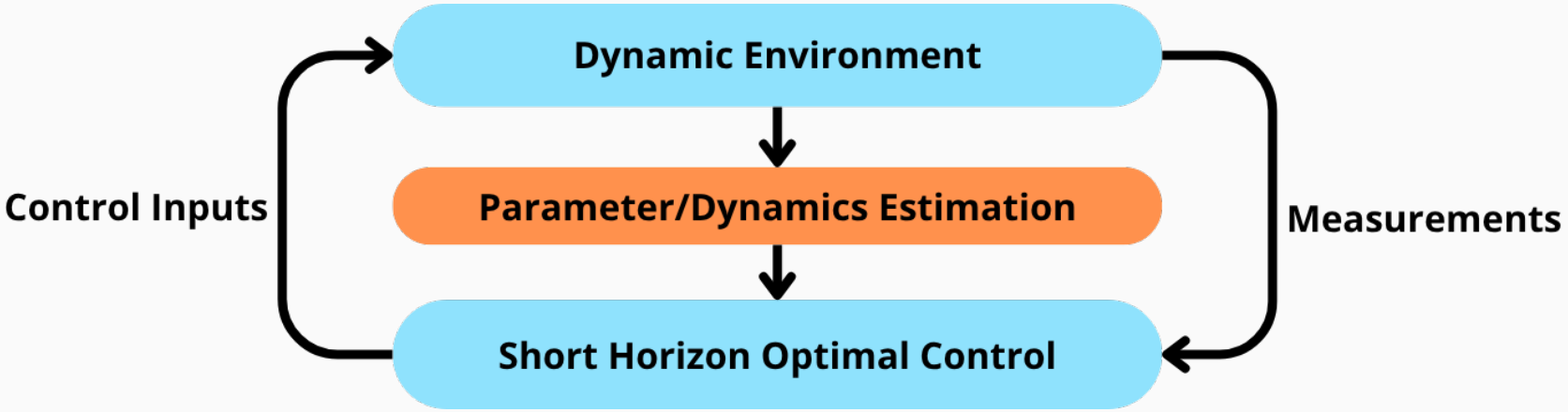
Special topics

- Model-based RL: learn f and r from data (or assume them known).
- Inverse RL: learn r from expert demonstrations.
- Multi-agent RL: multiple agents interacting in a shared environment.
- Meta-RL: learn to learn across multiple tasks/environments.
- Imitation learning: learn a policy from expert demonstrations without explicit rewards.

Model Predictive Control vs Reinforcement Learning

Data-driven MPC learns the model (and control) from data [8, 24]

Model-based RL plans against a learned model [1, 2].

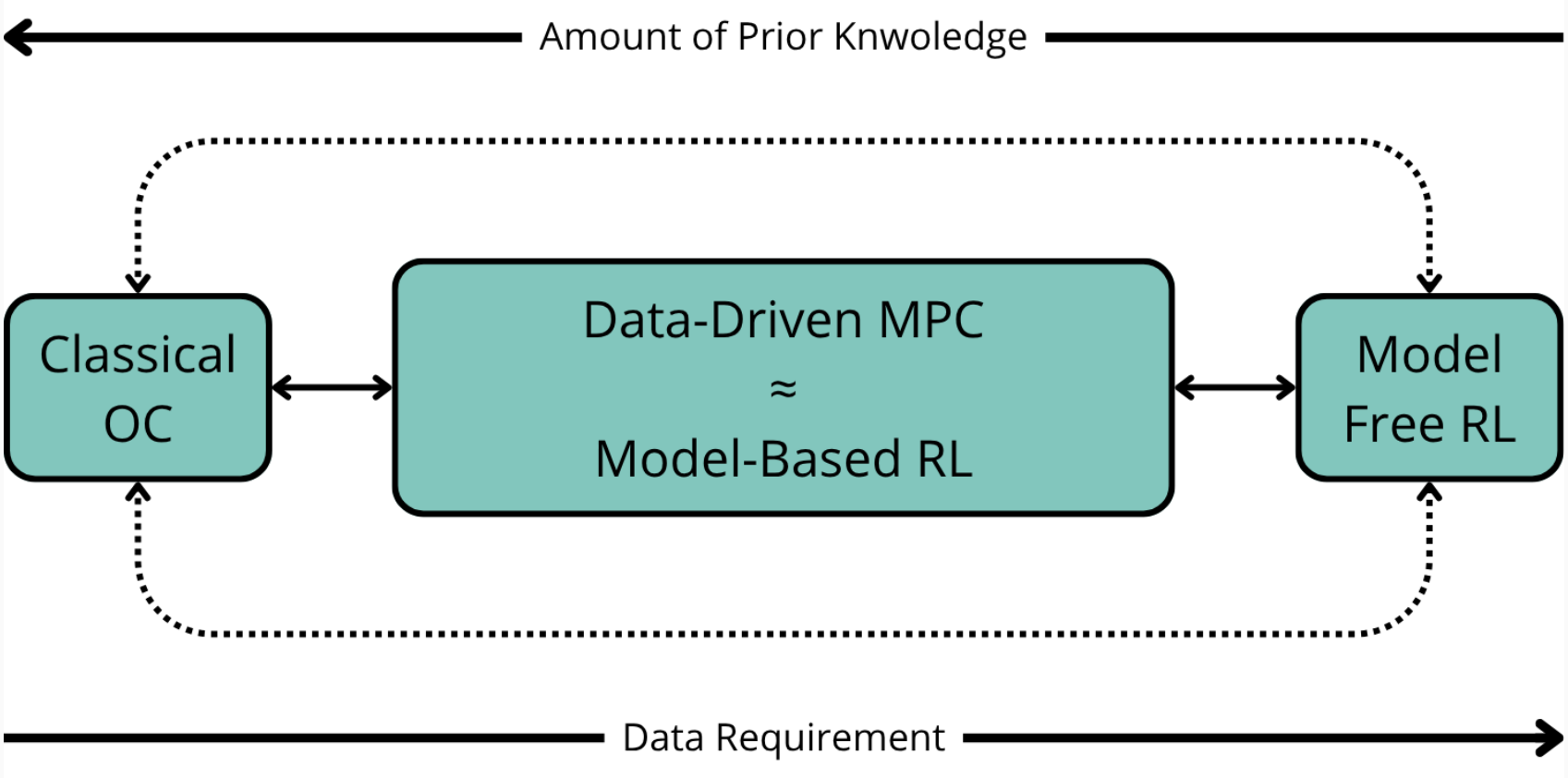


Source: Fredson Silva, MSc Student.

Model Predictive Control vs Reinforcement Learning

Data-driven MPC learns the model (and control) from data [8, 24]

Model-based RL plans against a learned model [1, 2].



Source: Fredson Silva, MSc Student.

Challenges of Reinforcement Learning

- Exploration versus exploitation. See Q-learning example.
- Sparse rewards. If the agent only gets a reward at the end of a long episode, it may never learn which actions were good or bad.
- Delayed rewards and long-horizon planning. Sometimes an agent only learns whether it did well much later. Which action actually caused success or failure?
- Sample inefficiency. Many RL methods require enormous numbers of interactions.
- Reward design and reward hacking. The behavior you get depends heavily on the reward function. If the reward is misspecified, the agent may exploit loopholes.
- Instability during training. Unlike in supervised learning, small changes in hyperparameters, random seeds, reward scaling, or environment dynamics can produce very different results.

Criticism of "Pure" Reinforcement Learning

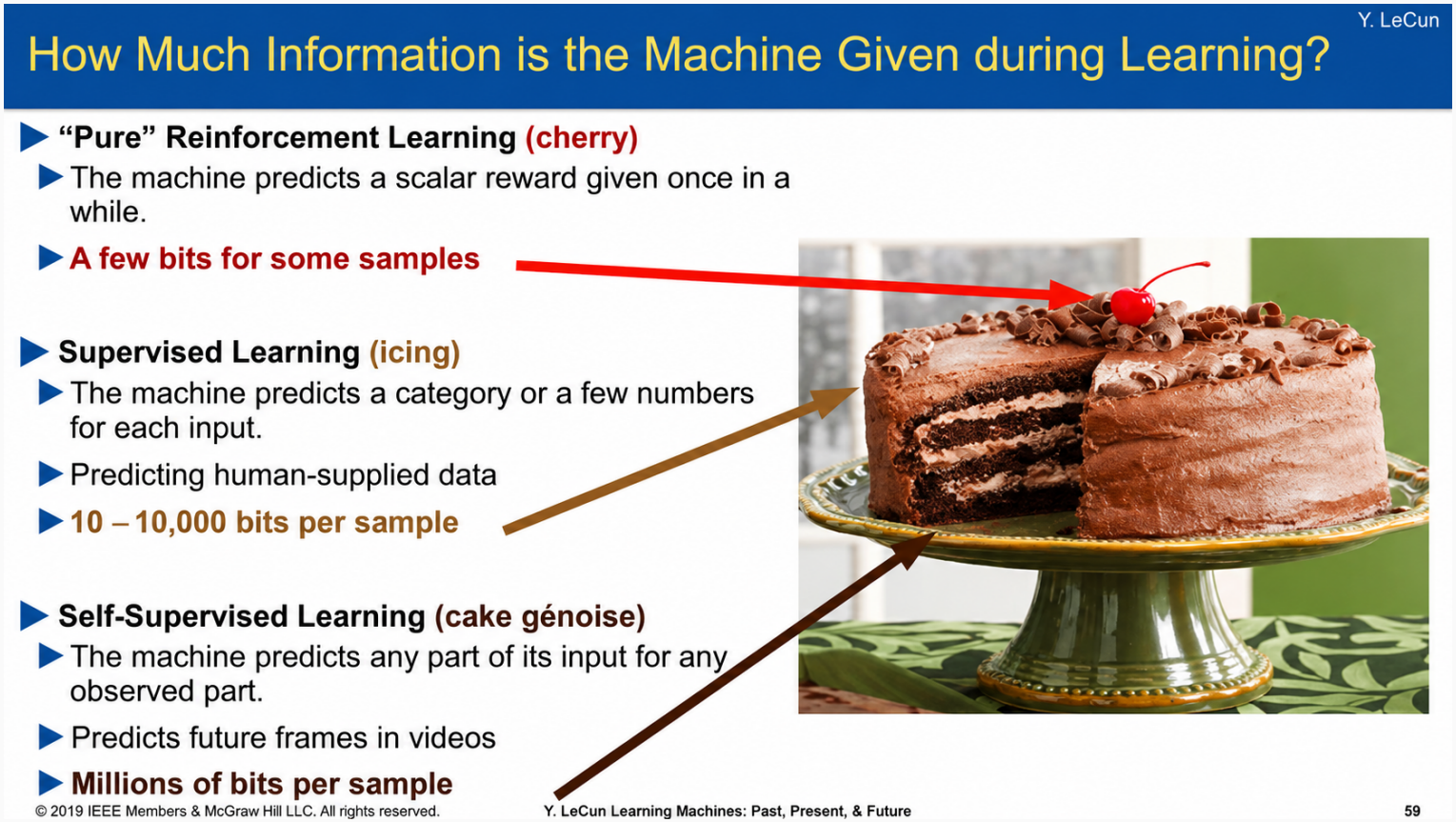


Figure 4: Yann LeCun, IEEE 2019

References

[1] Soroush Asri and Luis Rodrigues. Data-driven LQR using reinforcement learning and quadratic neural networks. *arXiv preprint arXiv:2311.10235*, 2023. URL `https://arxiv.org/abs/2311.10235`.

[2] Sarah Bechtle, Yixin Lin, Akshara Rai, Ludovic Righetti, and Franziska Meier. Curious iLQR: Resolving uncertainty in model-based RL. *arXiv preprint arXiv:1904.06786*, 2019. URL `https://arxiv.org/abs/1904.06786`.

[3] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952. doi: 10.1073/pnas.38.8.716.

[4] Richard E. Bellman. An introduction to the theory of dynamic programming. Technical Report R-245, RAND Corporation, Santa Monica, CA, 1953.

[5] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31*, pages 6571–6583, 2018.

[6] Giorgio Fabbri, Fausto Gozzi, and Andrzej Święch. *Stochastic Optimal Control in Infinite Dimension: Dynamic Programming and HJB Equations*. Probability Theory and Stochastic Modelling. Springer Cham, 2017. doi: 10.1007/978-3-319-53067-3.

References ii

[7] Farama Foundation. Humanoid - gymnasium documentation, 2024. URL https://gymnasium.farama.org/environments/mujoco/humanoid/. Opened on 08.01.2025.

[8] Armin Gießler, Felix Thömmes, and Sören Hohmann. Data-driven continuous-time linear quadratic regulator via closed-loop and reinforcement learning parameterizations. *arXiv preprint arXiv:2604.27922*, 2026. URL https://arxiv.org/abs/2604.27922.

[9] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290.

[10] Yijie Huang, Mengge Li, Xiang Yu, and Zhou Zhou. Continuous-time reinforcement learning for optimal switching over multiple regimes. *arXiv preprint arXiv:2512.04697*, 2025. URL https://arxiv.org/abs/2512.04697.

[11] Yanwei Jia and Xun Yu Zhou. q-learning in continuous time. *Journal of Machine Learning Research*, 24(161):1–61, 2023. URL https://www.jmlr.org/papers/volume24/22-0755/22-0755.pdf.

[12] Rudolf E. Kalman. Contributions to the theory of optimal control. *Boletin de la Sociedad Matematica Mexicana*, 5:102–119, 1960.

[13] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes*. Interscience Publishers, John Wiley & Sons, New York, 1962.

References iii

[14] Antonin Raffin and Freek Stulp. Generalized state-dependent exploration for deep reinforcement learning in robotics. *CoRR*, abs/2005.05719, 2020. URL `https://arxiv.org/abs/2005.05719`.

[15] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

[17] Wolfram Schultz, Peter Dayan, and P. Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997. doi: 10.1126/science.275.5306.1593.

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

[19] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063. MIT Press, 2000.

[20] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4): 279–292, 1992. doi: 10.1007/BF00992698.

[21] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696.

References iv

[22] Zhongqu Xie, Long Li, and Xiang Luo. A foot-ground interaction model based on contact stability optimization for legged robot. *Journal of Mechanical Science and Technology*, 36(2):921–932, 2022. doi: 10.1007/s12206-022-0139-1.

[23] Cagatay Yildiz, Markus Heinonen, and Harri Lähdesmäki. Continuous-time model-based reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139, pages 12009–12018, 2021. URL https://proceedings.mlr.press/v139/yildiz21a.html.

[24] Jian Zheng and Mario Sznaier. Robust data-driven receding-horizon control for LQR with input constraints. *arXiv preprint arXiv:2510.06117*, 2025. URL https://arxiv.org/abs/2510.06117.

Appendix

Deriving the adjoint (continuous & discrete)

Continuous case. Constraint $\dot{x}_t = f(x_t, u_t)$ enforced via a multiplier $p : [0, T] \to \mathbb{R}^n$:

$$\mathcal{L}(x, u, p) = \int_0^T \ell(x_t, u_t)\, dt + \ell_T(x_T) + \int_0^T p_t^\top \big(f(x_t, u_t) - \dot{x}_t\big)\, dt.$$

Integration by parts: $\int_0^T p^\top \dot{x}\, dt = p_T^\top x_T - p_0^\top x_0 - \int_0^T \dot{p}^\top x\, dt$. First variation ($\delta x_0 = 0$):

$$\delta\mathcal{L} = \int_0^T \big(\nabla_x \ell + (\nabla_x f)^\top p + \dot{p}\big)^\top \delta x\, dt + \int_0^T \big(\nabla_u \ell + (\nabla_u f)^\top p\big)^\top \delta u\, dt + \big(\nabla \ell_T(x_T) - p_T\big)^\top \delta x_T.$$

Cancelling the δx terms picks the adjoint dynamics; the δu term gives the gradient:

$$\boxed{\dot{p}_t = -\nabla_x \ell - (\nabla_x f)^\top p_t, \qquad p_T = \nabla \ell_T(x_T), \qquad \nabla_{u(t)} J = \nabla_u \ell + (\nabla_u f)^\top p_t.}$$

Discrete case. With $x_{t+1} = x_t + \Delta t\, f(x_t, u_t)$ and multipliers $\hat{p}_{t+1}$,

$$\mathcal{L} = \sum_{t=0}^{N-1} \Delta t\, \ell_t + \ell_T(x_N) + \sum_{t=0}^{N-1} \hat{p}_{t+1}^\top \big(x_t + \Delta t\, f_t - x_{t+1}\big).$$

$\partial_{x_t} \mathcal{L} = 0$ for $0 < t < N$ yields the backward sweep and $\partial_{u_t} \mathcal{L}$ gives the gradient:

$$\boxed{\hat{p}_t = \hat{p}_{t+1} + \Delta t \left(\nabla_x \ell_t + (\nabla_x f_t)^\top \hat{p}_{t+1}\right), \quad \hat{p}_N = \nabla \ell_T(x_N), \quad \nabla_{u_t} J = \Delta t \left(\nabla_u \ell_t + (\nabla_u f_t)^\top \hat{p}_{t+1}\right)}$$

This is exactly reverse-mode autodiff through the Euler unroll.

Policy gradient theorem: Proof (1/3)

Proof. Since $|G(\tau)| \le \|r\|_\infty/(1-\gamma)$, dominated convergence lets us differentiate under the integral, then use $\nabla_\theta p_\theta = p_\theta \, \nabla_\theta \log p_\theta$:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int G(\tau)\, p_\theta(\tau)\, d\tau \\ &= \int G(\tau)\, p_\theta(\tau)\, \nabla_\theta \log p_\theta(\tau)\, d\tau \;=\; \mathbb{E}_\tau[G(\tau)\, \nabla_\theta \log p_\theta(\tau)]\,.\end{aligned}$$

Take logs of the trajectory factorization and split by θ-dependence:

$$\log p_\theta(\tau) = \log \nu(S_0) + \sum_{t\ge 0} \log \pi_\theta(A_t \mid S_t) + \sum_{t\ge 0} \log P(S_{t+1} \mid S_t, A_t),$$

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t\ge 0} \nabla_\theta \log \pi_\theta(A_t \mid S_t) \qquad (\nu,\ P \text{ free of } \theta).$$

Substituting back:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_\tau\left[\sum_{t\ge 0} \nabla_\theta \log \pi_\theta(A_t \mid S_t) \;\cdot\; \sum_{k\ge 0} \gamma^k R_{k+1}\right]$$

Policy gradient theorem: Proof (2/3)

Step 2 (score is centered). For every fixed s and any θ,

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a \mid s)] = \int \pi_\theta(a \mid s)\, \frac{\nabla_\theta \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}\, da$$
$$= \nabla_\theta \int \pi_\theta(a \mid s)\, da \;=\; \nabla_\theta(1) \;=\; 0.$$

Step 3 (causality: past rewards drop out). Let $\mathcal{F}_t := \sigma(S_0, A_0, R_1, \ldots, R_t, S_t)$ — the history *up to but not including* A_t. Two facts from the construction of p_θ:

- For $k < t$: R_{k+1} is $\mathcal{F}_t$-measurable (since $k+1 \le t$).
- Conditionally on $\mathcal{F}_t$, $A_t \sim \pi_\theta(\cdot \mid S_t)$.

Tower property, then Step 2 applied with $s = S_t$:

$$\mathbb{E}[R_{k+1}\, \nabla_\theta \log \pi_\theta(A_t \mid S_t)] = \mathbb{E}\Big[R_{k+1}\, \underbrace{\mathbb{E}\big[\nabla_\theta \log \pi_\theta(A_t \mid S_t) \,\big|\, \mathcal{F}_t\big]}_{=0 \text{ (Step 2)}}\Big] \;=\; 0, \qquad (k < t).$$

So in the boxed formula only $k \ge t$ survives:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_\tau \left[\sum_{t\ge 0} \nabla_\theta \log \pi_\theta(A_t \mid S_t)\, \sum_{k\ge t} \gamma^k R_{k+1} \right].$$

Policy gradient theorem: Proof (3/3)

Step 4 (Markov property $\Rightarrow$ Q-function). Let $\mathcal{G}_t := \sigma(\mathcal{F}_t, A_t)$. The transition $P(S_{t+1} \mid S_t, A_t)$ and the subsequent rollout depend on history only through (S_t, A_t), so the conditional law of $(S_{t+1}, A_{t+1}, \dots) \mid \mathcal{G}_t$ equals that of a fresh rollout from (S_t, A_t):

$$\mathbb{E}\left[\sum_{k\geq t} \gamma^{k-t} R_{k+1} \,\middle|\, \mathcal{G}_t\right] \;=\; \mathbb{E}\left[\sum_{k\geq t} \gamma^{k-t} R_{k+1} \,\middle|\, S_t, A_t\right] \;=\; Q_{\pi_\theta}(S_t, A_t).$$

Step 5 (assemble). In each t-summand of Step 3, factor $\gamma^k = \gamma^t \gamma^{k-t}$ and apply the tower with $\mathcal{G}_t$; since $\nabla_\theta \log \pi_\theta(A_t \mid S_t)$ is $\mathcal{G}_t$-measurable, it pulls out:

$$\begin{aligned}\mathbb{E}\left[\nabla_\theta \log \pi_\theta(A_t \mid S_t) \sum_{k\geq t} \gamma^k R_{k+1}\right] &= \gamma^t\, \mathbb{E}\left[\nabla_\theta \log \pi_\theta(A_t \mid S_t)\, \mathbb{E}\Big[\sum_{k\geq t} \gamma^{k-t} R_{k+1} \,\Big|\, \mathcal{G}_t\Big]\right] \\ &= \gamma^t\, \mathbb{E}[\nabla_\theta \log \pi_\theta(A_t \mid S_t)\, Q_{\pi_\theta}(S_t, A_t)]\,.\end{aligned}$$

Summing over $t \geq 0$:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{t\geq 0} \gamma^t\, \nabla_\theta \log \pi_\theta(A_t \mid S_t)\; Q_{\pi_\theta}(S_t, A_t)\right]. \qquad \square$$