

# Deep Reinforcement Learning — Simon Hirländer

From Function Approximation to State-of-the-Art Continuous Control

Actor-Critic · DDPG · SAC & TD3 · Offline RL · Bridge to the Challenge

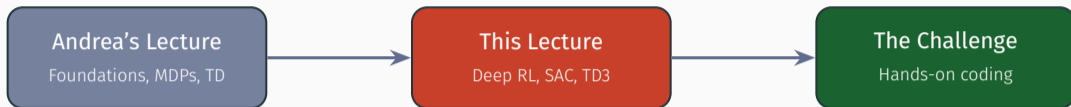
---

Simon Hirländer

RL4AA'26 — Liverpool, 30 March 2026

IDA Lab, Paris Lodron University of Salzburg

# Where We Are



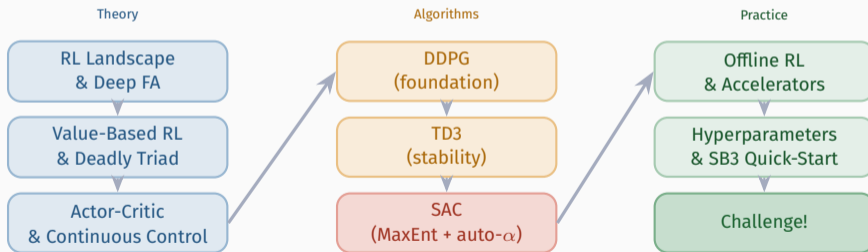
**You already know:** MDPs, value functions, Bellman equations, TD learning, Q-learning.

**Now:** How to scale RL to **continuous actions** and **high-dimensional** problems.

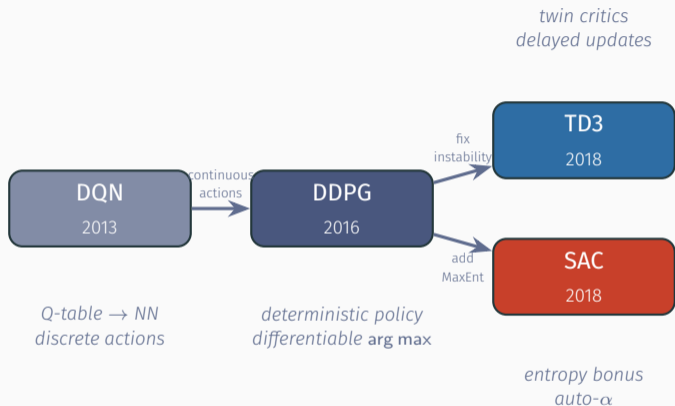
**Goal:** Understand SAC & TD3 well enough to use them in the challenge.

# Outline

**Goal:** understand the path from Q-learning to TD3 & SAC, and deploy them on a real accelerator control problem.



# Algorithm Family Tree: How We Got Here

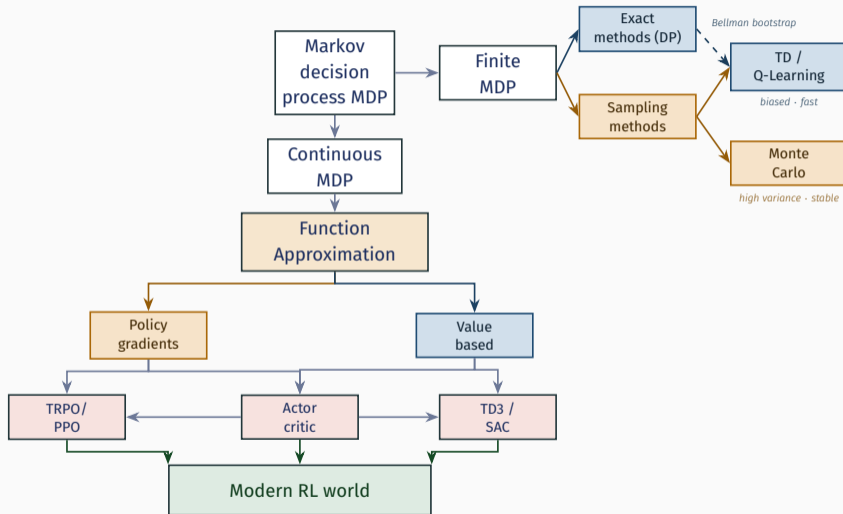


**Today's journey:** Each algorithm fixes a concrete flaw in its predecessor.  
By the end you will understand every arrow above.

# Overview: The RL Landscape

---

# A Map of Reinforcement Learning Methods



# Why Function Approximation (FA) in RL?

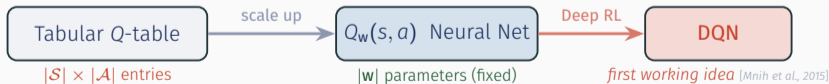
Tabular RL stores one value per state — fine for small problems, breaks for real ones.

## The Curse of Dimensionality

- Chess:  $\sim 10^{43}$  states
- Atari pixel obs:  $256^{100 \times 200 \times 3}$  states
- Accelerator: **continuous**  $s \in \mathbb{R}^n$   
⇒ infinitely many states
- A table simply **cannot exist**

## FA: The Fix

- Represent  $Q(s, a) \approx Q_w(s, a)$  with a **compact model**
- **Generalise** across similar states automatically
- Same number of parameters  $|w|$  regardless of  $|S|$
- Neural networks ⇒ **Deep RL**



# Why replacing the Q-table with a FA is not sufficient

DQN [Mnih et al., 2015] works great for:

- Discrete actions (Atari games, grid worlds)
- $\arg \max_a Q(s, a)$  is easy when  $|\mathcal{A}|$  is finite

But in accelerator control:

- Actions are **continuous** (magnet currents, RF phases)
- $\arg \max_a Q(s, a)$  requires solving an optimization at every step
- We may want **stochastic** policies for exploration

Discrete



Continuous



**Solution:** Learn the policy **directly** — *Policy Gradient* methods, or scale Q-learning with **function approximation**.

## Parametric (fixed-size $w$ )

- **Linear:**  $\hat{V}(s) = w^\top \phi(s)$  – converg. guarantees
- **Tile Coding / RBF** – hand-crafted features
- **MLP / CNN / RNN** – universal approx., GPU
- **Transformer** – sequences, multi-agent

## Non-Parametric (grows with data)

- **Gaussian Processes** – uncertainty, poor scaling
- **$k$ -Nearest Neighbours** – memory-based, no training
- **Decision Trees / Forests** – interpretable, discrete
- **Kernel Methods (SVM)** –  $\mathcal{O}(n^2)$  cost

## Why Parametric (Deep) Wins in RL

- Fixed memory; end-to-end gradient descent; scales with compute
- **Cost:** no convergence guarantee with bootstrapping + off-policy (*Deadly Triad*)

# Why is this a problem: The Bellman Operator and the Contraction Property

Define the Bellman optimality operator  $\mathcal{T}^*$ :

$$(\mathcal{T}^*Q)(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$$

## Contraction Mapping Theorem (Banach)

$\mathcal{T}^*$  is a  $\gamma$ -contraction in the  $\ell_\infty$  norm:

$$\|\mathcal{T}^*Q_1 - \mathcal{T}^*Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty, \quad \gamma \in (0, 1)$$

**Consequence:** Repeated application converges to the *unique* fixed point  $Q^*$ :

$$Q^* = \mathcal{T}^*Q^* \implies \text{Value iteration} = \text{guaranteed convergence!}$$

- Tabular: each Bellman update is **exact**  $\rightarrow$  contraction is preserved
- Q-learning is a *stochastic* version of value iteration — still converges [Watkins & Dayan, 1992]
- Convergence rate:  $\gamma^k$  at step  $k$  — the lower  $\gamma$ , the faster

# Function Approximation Breaks Convergence

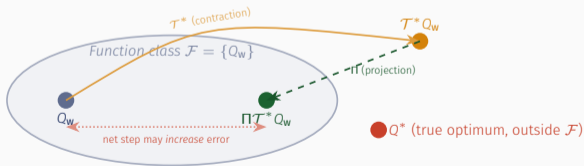
With FA we cannot store  $Q^*$  exactly. Each update becomes:

$$Q_w \leftarrow \Pi \mathcal{T}^* Q_w$$

where  $\Pi$  = **projection** onto the function class  $\{Q_w : w \in \mathbb{R}^d\}$ .

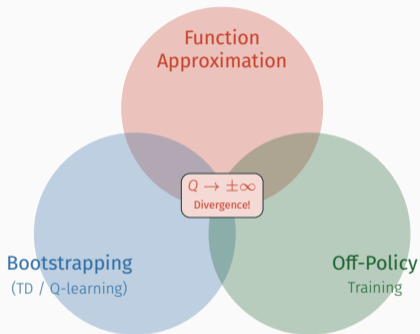
## The Problem: $\Pi \mathcal{T}^*$ is NOT a contraction

- Projection can undo (or overshoot) the contraction step
- Errors may **accumulate** across iterations; no fixed-point theorem applies
- Divergence is possible even for simple linear FA [Baird, 1995]



# The Deadly Triad

Sutton & Barto (2018) [Sutton & Barto, 2018]: instability or divergence arises when **all three** are combined.



**Any two alone:** manageable.

**All three together:** Q-values can diverge.

**Practical mitigations:**

- Target networks (freeze critic targets)
- Double Q (clipped Twin critics)
- Replay buffer (decorrelates data)
- Gradient clipping

These help in practice but provide *no* theoretical convergence guarantee.

**Alternative:** skip value estimation entirely — learn the policy **directly**.

No bootstrapping ⇒ no deadly triad ⇒ **Policy Gradient methods**.

# Value-Based RL: Key Characteristics

## Strengths

- **Solid theoretical basis:** unique  $Q^*$  via Bellman optimality
- **Tabular convergence:** Q-learning reaches  $Q^*$  [Watkins & Dayan, 1992]
- **Off-policy:** replay buffer reuses past experience
- **Fast credit assignment:** TD bootstrap, no full episodes needed
- **Explicit action values:** greedy policy readable from  $Q$
- **Proven at scale:** DQN — superhuman on 57 Atari games [Mnih et al., 2015]

## Limitations

- **Continuous actions:**  $\arg \max_a Q(s, a)$  is an optimisation at every step
- **Deadly Triad:** FA + bootstrap + off-policy  $\Rightarrow$  no convergence guarantee
- **Q-value divergence:** possible even for linear FA [Baird, 1995]
- **Heuristic fixes only:** target nets, double-Q, replay — no theoretical guarantee
- **Bootstrap bias:** max-operator overestimates  $Q$
- **No native stochastic policy:** greedy  $\Rightarrow$  poor exploration

## Bottom line & bridge

Value-based RL is **sample-efficient and theoretically grounded**, but the Deadly Triad makes deep value-based learning **fragile in practice**. **Alternative:** learn  $\pi_\theta(a|s)$  directly  $\Rightarrow$  **Policy Gradient methods** — no bootstrapping, native continuous & stochastic actions.

# Policy-Based RL: Key Characteristics

## Strengths

- **Continuous actions:** maps  $s \rightarrow a$  natively; no intractable  $\arg \max_a$  solver needed
- **Stochastic policies:** natural exploration and ability to learn optimal stochastic behaviour
- **Direct optimisation:** follows the true gradient of the objective  $J(\theta)$
- **Convergence:** guaranteed to reach at least a local optimum (smooth updates)
- **Stability:** pure policy gradients do not bootstrap  $\Rightarrow$  no Deadly Triad

## Limitations

- **Sample inefficiency:** purely on-policy; learns from data once, then discards it
- **High variance:** Monte Carlo returns lead to noisy gradient estimates
- **Local optima:** gradient ascent can get trapped
- **Slow learning:** requires full episodes, no fast TD credit assignment

## Bottom line & bridge

Policy-based RL is **stable and handles continuous actions elegantly**, but purely on-policy updates make it **too sample-inefficient** for many real-world tasks. **Alternative:** combine the sample-efficiency of value functions with the continuous action capability of policy methods  $\Rightarrow$  **Actor-Critic methods**.

## The Actor: Parameterised Policy

---

# The Actor: Parameterised Policy

The **actor** is a neural network that maps states directly to actions — replacing the intractable  $\arg \max$  with a single forward pass.

## Policy as a Neural Network

- **Input:** state  $s$  (e.g. BPM readings)
- **Output:** action  $a$  (e.g. corrector strengths)
- **Parameters:**  $\theta$  trained by gradient ascent on  $Q_\phi(s, a)$

Two flavours used in this lecture:

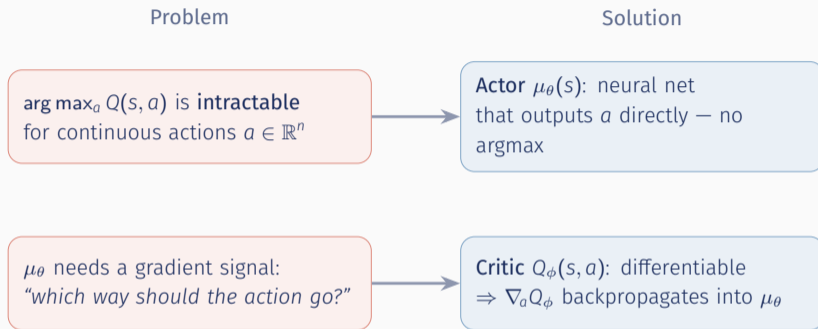
$$\mu_\theta(s) \quad (\text{deterministic — TD3}) \quad \pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)) \quad (\text{stochastic — SAC})$$

The critic  $Q_\phi(s, a)$  tells the actor *how good* each action is. Together they form the **Actor-Critic** architecture.

## Why Actor-Critic?

---

# Why Actor-Critic?



Actor-Critic = differentiable argmax trained by a differentiable value function.

→ This is exactly what DDPG, TD3, and SAC do.

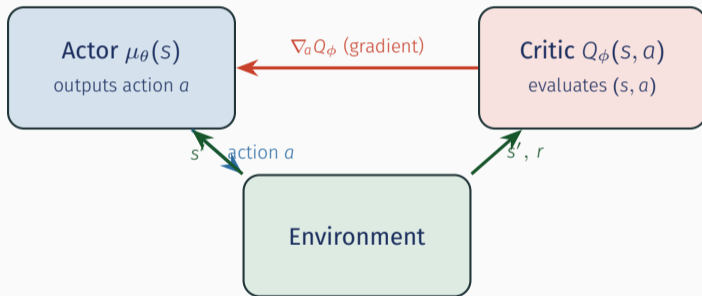
## Actor-Critic & Off-Policy Learning

---

# The Actor-Critic Architecture — Best of Both Worlds

**Key insight:** the critic  $Q_\phi(s, a)$  is **differentiable w.r.t.  $a$** . Its gradient trains the actor directly — no Monte Carlo rollouts needed:

$$\nabla_{\theta} J \approx \mathbb{E} \left[ \nabla_a Q_\phi(s, a) \Big|_{a=\mu_\theta(s)} \cdot \nabla_{\theta} \mu_\theta(s) \right]$$



- **Actor** ( $\mu_\theta$ ): selects actions — the differentiable argmax
- **Critic** ( $Q_\phi$ ): evaluates  $(s, a)$ , backpropagates  $\nabla_a Q_\phi$  into actor

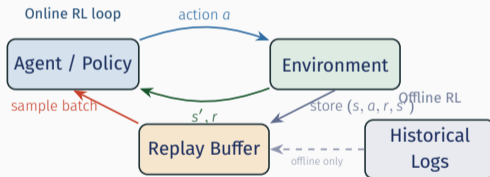
# On-Policy vs. Off-Policy

## On-Policy [Schulman et al., 2017] (e.g. PPO)

- Learns from data collected by *current* policy
- Data used once, then discarded
- Stable but **sample inefficient**
- Needs millions of environment steps

## Off-Policy (e.g. SAC, TD3)

- Learns from *any* past data (replay buffer)
- Data reused many times
- **Much more sample efficient**
- Can learn from demonstrations



**Key enabler:** The **replay buffer** stores transitions and allows reuse.

In **offline RL** (coming soon) the buffer is filled from logs — no live environment needed.

## DDPG — The Foundation

---

# DDPG: Deep Deterministic Policy Gradient

DDPG [Lillicrap et al., 2016] = DQN ideas + Actor-Critic for continuous actions.

## Solving the $\arg \max$ problem for continuous actions

DQN needs  $\arg \max_a Q(s, a)$  — intractable when  $a \in \mathbb{R}^n$ .

DDPG learns a **deterministic policy**  $\mu_\theta(s)$  so that:

$$\max_a Q(s, a) \approx Q(s, \mu_\theta(s))$$

The policy becomes the *differentiable*  $\arg \max$ .

## Key ingredients:

1. **Deterministic policy:**  $a = \mu_\theta(s)$
2. **Critic:**  $Q_\phi(s, a)$  trained with Bellman error
3. **Replay buffer:** off-policy data reuse
4. **Target networks:** slow-moving copies

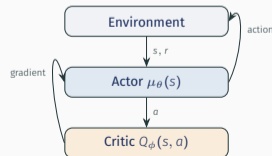
$$\phi' \leftarrow \tau \phi + (1-\tau) \phi'$$

5. **Exploration:** Gaussian noise on actions

$$a = \mu_\theta(s) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

## The Key Insight

The policy *becomes* a differentiable  $\arg \max$ :  
a beautiful idea all modern algorithms inherit.



## But DDPG Has Problems...

- **Q-function overestimates values** — max-operator bias compounds over training
- **Brittle to hyperparameters** — learning rate, noise  $\sigma$ ,  $\tau$  all require tuning
- **Exploration noise is manual** —  $\mathcal{N}(0, \sigma)$  schedule must be set by hand
- **Often unstable in practice** — training can collapse without warning

**TD3** and **SAC** keep the idea — fix the problems.

## TD3 — Twin Delayed DDPG

---

# TD3: Three Tricks to Fix DDPG

TD3 [Fujimoto et al., 2018] — three targeted fixes:

## 1. Twin Q-Networks (Clipped Double Q-Learning)

- Train *two* critics  $Q_{\phi_1}, Q_{\phi_2}$
- Use the **minimum** for the target:  $y = r + \gamma \min_i Q_{\phi'_i}(s', \tilde{a}')$
- Prevents overestimation bias

## 2. Delayed Policy Updates

- Update actor every  $d$  critic updates (typically  $d = 2$ )
- Lets the critic stabilise before the actor chases it

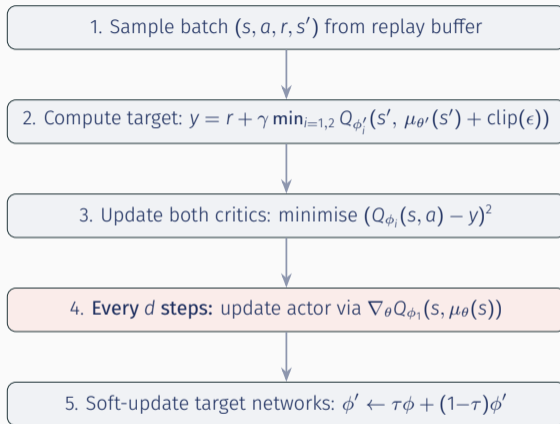
## 3. Target Policy Smoothing

- Add clipped noise to target actions:

$$\tilde{a}' = \mu_{\theta'}(s') + \text{clip}(\epsilon, -c, c), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

- Regularises the critic, prevents sharp Q-peaks

## TD3: Algorithm Overview



Deterministic policy → exploration via **explicit noise** added during data collection.

## SAC — Soft Actor-Critic

---

# Control as Inference: The Probabilistic Foundation of SAC

Optimality Variable  $\mathcal{O}_t \in \{0, 1\}$

$$p(\mathcal{O}_t=1 | s_t, a_t) \propto \exp(r(s_t, a_t))$$

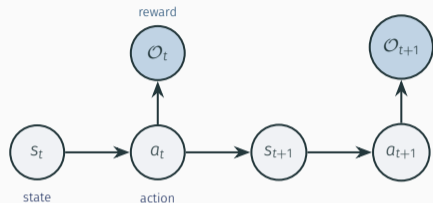
**Goal:** compute the posterior policy

$$\pi^*(a_t | s_t) = p(a_t | s_t, \mathcal{O}_{t:\infty}=1)$$

Variational Objective — minimise KL divergence

$$\max_{\pi} \mathbb{E}[\sum_t r_t] + \alpha \mathbb{E}[\sum_t \mathcal{H}(\pi(\cdot | s_t))]$$

Entropy term  $\mathcal{H}$  emerges naturally from the KL, with temperature  $\alpha$ .



Shaded nodes observed as optimal.

Solving this posterior *exactly* yields the MaxEnt RL objective.

# SAC: The Maximum Entropy Framework

SAC [Haarnoja et al., 2018] — key insight: maximise reward AND entropy.

$$J(\theta) = \sum_t \mathbb{E} \left[ r_t + \alpha \mathcal{H}(\pi_{\theta}(\cdot | S_t)) \right]$$

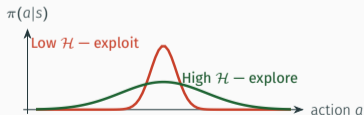
where  $\mathcal{H}(\pi) = -\mathbb{E}[\log \pi(a|s)]$  is the entropy of the policy.

## Why entropy matters:

- Automatic exploration — no manual noise
- More robust policies
- Better multi-modal behaviour
- Prevents premature convergence

## $\alpha$ — the temperature:

- Controls exploration–exploitation trade-off
- $\alpha$  large  $\rightarrow$  more exploration
- $\alpha$  small  $\rightarrow$  more exploitation
- SAC auto-tunes  $\alpha$ !



# SAC: Automatic Temperature Tuning

The killer feature: SAC automatically adjusts  $\alpha$  to maintain a target entropy.

$$\alpha^* = \arg \min_{\alpha} \mathbb{E}_{a \sim \pi} [ -\alpha \log \pi(a|s) - \alpha \bar{\mathcal{H}} ]$$

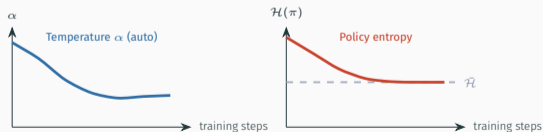
where  $\bar{\mathcal{H}}$  is the **target entropy** (typically  $= -\dim(\mathcal{A})$ ).

## Early training:

- Policy is uncertain  $\rightarrow$  entropy is high
- $\alpha$  decreases to match target
- Agent naturally explores

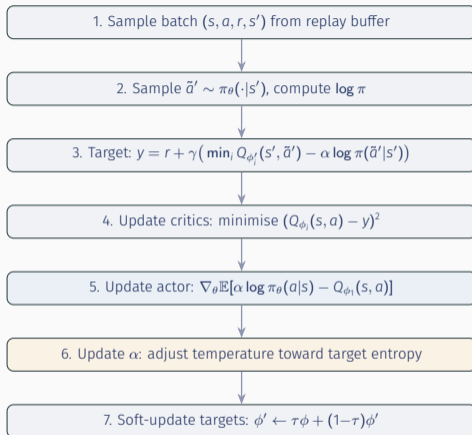
## Late training:

- Policy converges  $\rightarrow$  entropy drops
- $\alpha$  increases to prevent collapse
- Maintains healthy exploration



$\rightarrow$  Always use automatic  $\alpha$ . One less hyperparameter to tune!

# SAC: Algorithm Overview



## SAC vs. TD3 — When to Use What

---

# SAC vs. TD3: Head-to-Head

Property	TD3	SAC
Policy	Deterministic	Stochastic
Exploration	Gaussian noise (manual)	Entropy (automatic)
Q-networks	Twin (min)	Twin (min)
Temperature $\alpha$	—	Auto-tuned
Policy updates	Delayed ( $d = 2$ )	Every step
Target smoothing	Yes (noise)	Yes (entropy)
Hyperparameter sensitivity	Medium	Low
Sample efficiency	High	High
Robustness	Good	Better

**Rule of thumb:** Start with **SAC**. Try TD3 if you need simpler debugging or lower compute.

# SAC vs. TD3: Schematic Performance Across Seeds

**Limitation:** Both still require **online interaction** — impossible when beam time is scarce.  $\Rightarrow$  Next section: learn from historical data, without touching the machine.



SAC's **narrower band** means more **reliable** results across random seeds.  
Both converge to similar final performance — SAC simply gets there more **consistently**.

# Offline Reinforcement Learning

---

# Offline RL: Learning Without Interaction

**Problem:** Online RL needs many environment interactions — expensive & risky!



*No new interactions!*

## Key Challenge: Distributional Shift

The policy may choose actions **never seen** in the dataset → Q-values become unreliable.

**Solutions:** Conservative Q-Learning (CQL), IQL, or **world-model approaches** (our work).

# Offline RL: Why It Matters for Accelerators

- Accelerators generate **terabytes** of logged operations data
- Beam time is **precious** — cannot afford millions of online RL steps
- Safety-critical: must avoid catastrophic actions during learning

## Our approach (Koopman-Stabilised World Models):

1. Archive data from simulations or operations
2. Learn a **stable surrogate model** (Koopman operator)
3. Train RL policies **entirely offline** on the surrogate
4. Use **epistemic uncertainty** to detect out-of-distribution states

→ Policies match online PPO performance **without any machine interaction**.

**Tomorrow morning (Tue 09:15): Keynote by Samuele Tosatto**

*“Accelerating RL with Off-Policy Data”* — going deeper on exactly these challenges!

# Practical Guide: Hyperparameters & Tips

---

# Hyperparameter Cheat Sheet

Hyperparameter	SAC	TD3	Advice
Learning rate (actor & critic)	3e-4	3e-4	Start here (Adam)
Replay buffer size	1e6	1e6	Bigger is better
Batch size	256	256	128–512, rarely critical
$\tau$ (soft update)	0.005	0.005	Don't touch
$\gamma$ (discount)	0.99	0.99	Lower for short horizons
$\alpha$ (entropy)	auto	—	Always use auto
Policy delay	1	2	TD3-specific
Exploration noise $\sigma$	—	0.1	TD3 only
Hidden layers	[256, 256]	[256, 256]	Good default

Start with defaults. Change one thing at a time.

# Top Practical Tips

1. **Normalise observations** — the single biggest improvement
  - Use running mean/std or fixed normalisation from your data
2. **Reward shaping matters** — scale rewards to roughly  $[-1, 1]$ 
  - SAC's entropy interacts with reward scale
3. **Action scaling** — ensure actions map to  $[-1, 1]$  internally
  - Rescale to physical units outside the algorithm
4. **Monitor Q-values** — if they diverge, something is wrong
  - Common sign of reward scaling or learning rate issues
5. **Seed averaging** — always run 3–5 seeds, report mean  $\pm$  std
  - RL results are notoriously noisy across seeds

# Quick Start: SAC & TD3 with Stable-Baselines3

## SAC (recommended default)

```
import gymnasium as gym
from stable_baselines3 import SAC

env = gym.make("Pendulum-v1")

model = SAC(
    "MlpPolicy", env,
    learning_rate=3e-4,
    buffer_size=1_000_000,
    batch_size=256,
    tau=0.005,
    gamma=0.99,
    ent_coef="auto", # auto!
    verbose=1,
)
model.learn(100_000)
```

## TD3 (if you prefer deterministic)

```
import gymnasium as gym
from stable_baselines3 import TD3

env = gym.make("Pendulum-v1")

model = TD3(
    "MlpPolicy", env,
    learning_rate=3e-4,
    buffer_size=1_000_000,
    batch_size=256,
    tau=0.005,
    gamma=0.99,
    policy_delay=2, # every 2 steps
    verbose=1,
)
model.learn(100_000)
```

Evaluate: `action, _ = model.predict(obs, deterministic=True)` – same API for both!

## Summary & Bridge to the Challenge

---

# What You've Learned

## Foundations

- **Why FA?** Tables break for continuous/high-dim spaces
- **Bellman + FA** = no convergence guarantee (Deadly Triad)
- **Policy Gradients:** learn  $\pi_\theta$  directly via  $\nabla_\theta \log \pi \cdot G_t$
- **Actor-Critic:** advantage  $A(s, a)$  cuts variance; off-policy replay cuts cost

## State-of-the-Art Algorithms

- **DDPG:** deterministic policy = differentiable arg max
- **TD3:** twin critics + delayed updates + target smoothing
- **SAC:** MaxEnt + auto- $\alpha$  = most robust default
- **Offline RL:** learn from logs, no machine interaction

## Decision rule for the challenge

Start with SAC · normalise observations · scale rewards to  $[-1, 1]$  · run 3 seeds

## Questions?

Simon Hirländer  
simon.hirlaender@plus.ac.at  
IDA Lab, University of Salzburg

**Next up: Challenge Group Assignment**

# References

## Core papers:

- Watkins & Dayan (1992). **Q-learning**. *Machine Learning*.
- Williams (1992). **REINFORCE**. *Machine Learning*.
- Baird (1995). **Residual algorithms**. *ICML*.
- Sutton et al. (1999). **Policy gradient theorem**. *NeurIPS*.
- Mnih et al. (2015). **DQN**. *Nature* 518.
- Lillicrap et al. (2016). **DDPG**. *ICLR*.
- Schulman et al. (2017). **PPO**. *arXiv:1707.06347*.
- Fujimoto et al. (2018). **TD3**. *ICML*.
- Haarnoja et al. (2018). **SAC**. *ICML*.
- Sutton & Barto (2018). **RL: An Introduction**. MIT Press.

## Further reading:

- Kumar et al. (2020). **CQL**. *NeurIPS*.
- Chen et al. (2021). **Decision Transformer**. *NeurIPS*.
- Chen et al. (2021). **REDQ**. *ICLR*.
- Kostrikov et al. (2021). **IQL**. *ICLR 2022*.
- Kuznetsov et al. (2021). **TQC**. *ICML*.
- Ouyang et al. (2022). **InstructGPT/RLHF**. *NeurIPS*.
- Hafner et al. (2023). **DreamerV3**. *arXiv:2301.04104*.
- Hansen et al. (2024). **TD-MPC2**. *ICLR*.
- Willi et al. (2024). **CrossQ**. *ICLR*.

## Backup Slides

Not part of the main talk — available if questions arise

# Deep RL Since 2018: What Has Changed?

## Sample Efficiency & Stability

- **REDQ** (2021): ensemble of  $N$  critics  
Massively improves sample efficiency; random subset for updates
- **CrossQ** (2024): batch-norm trick  
SAC-level performance with far fewer gradient steps
- **TQC** (2021): truncated quantile critics  
Distributional RL; better uncertainty; top SB3 benchmark

## Model-Based RL (MBRL)

- **DreamerV3** (2023): world model in latent space  
One algorithm, no tuning, across domains (Atari, DMC, Minecraft)
- **TD-MPC2** (2024): MPC + implicit world model  
State-of-the-art on continuous control, scales to 1B params

## Offline & Foundation RL

- **CQL** (2020): conservative Q-learning  
Penalises OOD actions; seminal offline RL paper
- **IQL** (2021): implicit Q-learning  
Avoids OOD queries entirely; simpler & often better
- **Decision Transformer** (2021): RL as sequence modelling  
GPT-style; conditions on return-to-go tokens
- **RLHF** (2022+): RL from human feedback  
Powers ChatGPT, Gemini; reward model from preferences

## The Big Shift

- **LLM + RL**: agents that plan with language (ReAct, GRPO)
- **Diffusion policies**: multi-modal, contact-rich control

# Algorithms Worth Exploring Next

## If you liked SAC / TD3...

1. **TQC** — drop-in upgrade, often outperforms SAC  
`pip install sb3-contrib`
2. **CrossQ** — same quality, 5× faster training  
Batch norm in critic; minimal code change
3. **REDQ** —  $N$ -ensemble critics for hard exploration  
Tune  $N$  and UTD ratio
4. **SAC-N** — push ensemble to  $N=500$  critics  
Offline RL *without* explicit conservatism

## For Accelerator / Safety Problems

1. **MBPO / DreamerV3** — model-based; few real steps
2. **TD-MPC2** — MPC + learned model; interpretable
3. **IQL / CQL** — offline RL from logs
4. **BEAR / MOPO** — offline RL with uncertainty

## If you want to go deeper...

1. **PPO + RND** — curiosity-driven on-policy exploration
2. **TRPO** — trust-region; theoretically grounded PPO ancestor
3. **MPO** (DeepMind) — maximum a-posteriori policy optimisation
4. **Diffusion Policies** — multi-modal; robotics SOTA
5. **Decision Transformer** — offline RL without Bellman

## Starting Points

- **Stable-Baselines3 + sb3-contrib** — most algorithms above
- **CleanRL** — single-file, readable implementations
- **CORL** — offline RL benchmark suite
- **Gymnasium / Shimmy** — environment wrappers

# Backup: Policy Gradient Theorem

**Objective:** Maximise expected return

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \gamma^t r_t \right]$$

**The Policy Gradient Theorem** [Sutton et al., 1999]:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | S_t) G_t \right]$$

where  $G_t = \sum_{k=t}^T \gamma^{k-t} r_k$  is the return-to-go.

**Intuition:**

- Actions that led to **high return** → **increase probability**
- Actions that led to **low return** → **decrease probability**

# Backup: REINFORCE and Its Problems

**REINFORCE** [Williams, 1992]: simplest policy gradient algorithm.

1. Collect a full episode using  $\pi_\theta$
2. Compute returns  $G_t$
3. Update:  $\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$

## The Variance Problem

- MC returns  $G_t$  are **noisy** — high variance
- Needs many episodes for stable gradients
- **On-policy**: data discarded after 1 use

**Fix:** replace  $G_t$  with critic estimate  $Q_\phi(s, a) \rightarrow$  **Actor-Critic**.

# Backup: SAC — Stochastic Policy with Squashing

Policy outputs (reparameterisation trick):

$$\mu_{\theta}(s), \sigma_{\theta}(s) \xrightarrow{\epsilon \sim \mathcal{N}(0, I)} z = \mu_{\theta}(s) + \sigma_{\theta}(s) \odot \epsilon \xrightarrow{\tanh} a = \tanh(z)$$

- **Reparameterisation**: gradients flow through  $\mu_{\theta}, \sigma_{\theta}$  (not through sample)
- The **tanh squashes** actions to  $[-1, 1]$  (bounded action space)
- Requires a **log-probability correction** (change of variables):

$$\log \pi(a|s) = \log \mathcal{N}(z|\mu, \sigma) - \sum_i \log(1 - \tanh^2(z_i))$$

## Key Advantage over TD3

- Exploration is **built into the policy** via entropy — no manual noise schedule
- Stochastic  $\rightarrow$  can represent multi-modal action distributions