Politecnico di Milano 1863
Scuola di Ingegneria industriale e dell'informazione

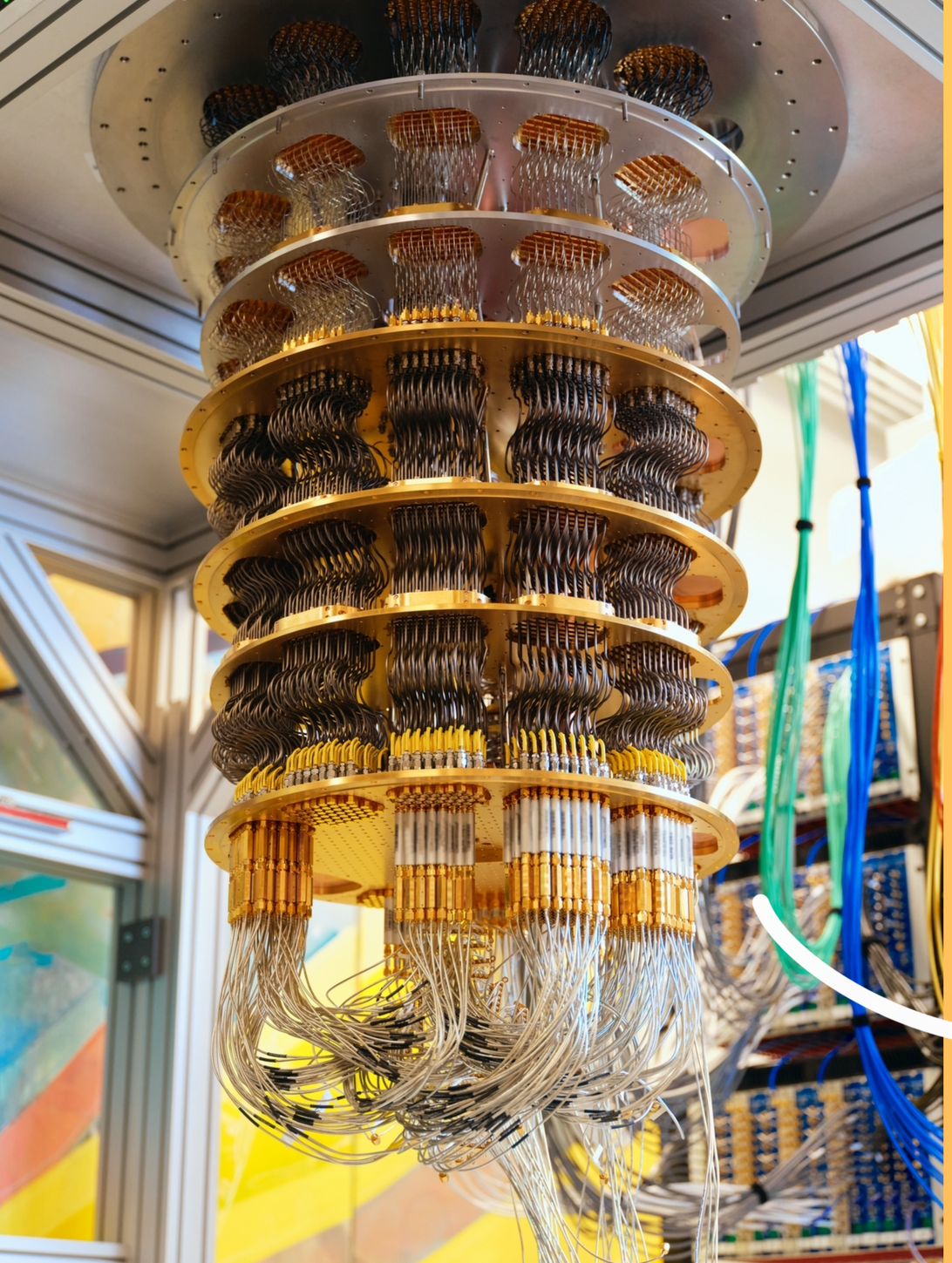
Reinforcement learning for Quantum optimal control

Alessio Parato

Advisor: Prof. Elisabetta Di Nitto

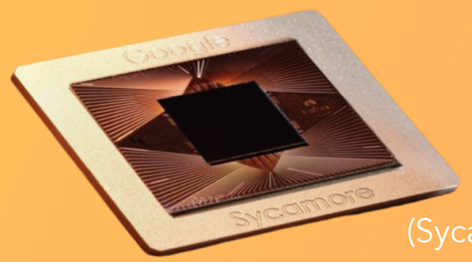
Co-advisors: Simone Reale, Patrick Hopf, Alexander Pitchford

Academic Year: 2023-24



The future of computing is Quantum

- Google’s Sycamore Quantum computer
  - The largest simulation in chemistry
  - Can tackle problems that classical machines find nearly impossible
- QOC (Quantum Optimal Control)
  - Manipulating qubits to control their quantum states
  - Finding parameters for control fields
  - Fidelity



(Sycamore quantum processors)

- The control landscape describes how the control objective changes with variations in the applied field parameters.
- Simple quantum systems typically lack local minima, but may have saddle points.
- Constraints (e.g., control duration, field intensity, discretization) complicate the landscape.
- These constraints can introduce traps, requiring robust optimization algorithms to avoid them.

QOC algorithms

$$H(t) = H_0 + \sum_j u_j(t) H_j$$

Drift H σ_z — Control fx. — Control H σ_x σ_y

Numerical solver

mesovle()

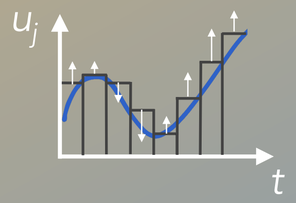
$$i\hbar \frac{\partial \psi}{\partial t} = H\psi$$

Schrödinger equation / Lindbland master equation

GRAPE

Discretized time slots

1° and 2° order ∇



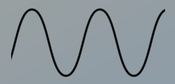
JOPT

Automatic differentiation

JAX

GOAT

Continuous analytical fx.



Gradient based optimization

CRAB

$$u(t) = \sum_n a_n sin(w_n t) + b_n cos(w_n t)$$

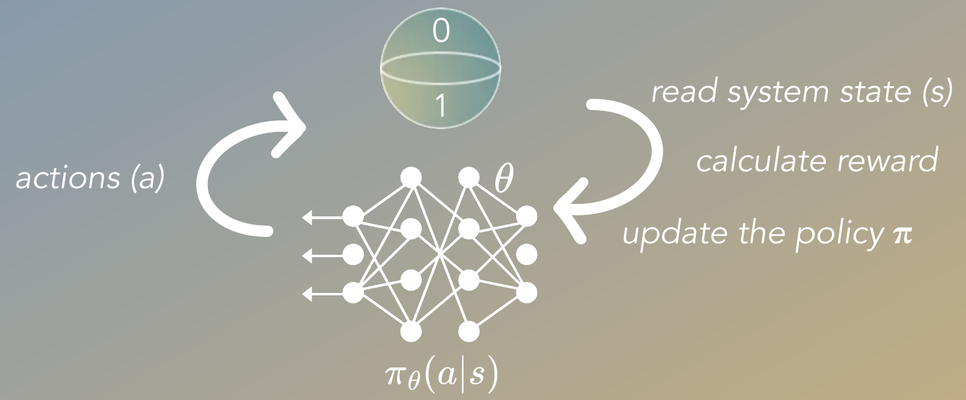
Random basis & Nelder-Mead

RL algorithm & Grid World example

RL algorithm characteristics

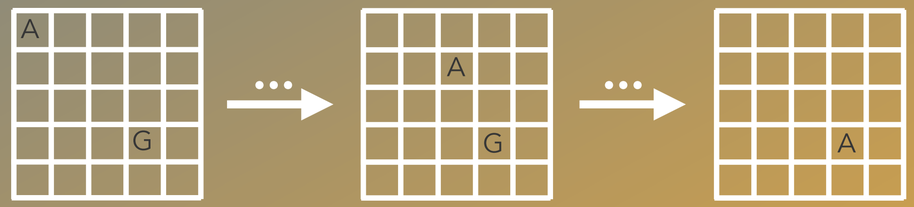
- Model-based (Hamiltonian)
- Update:

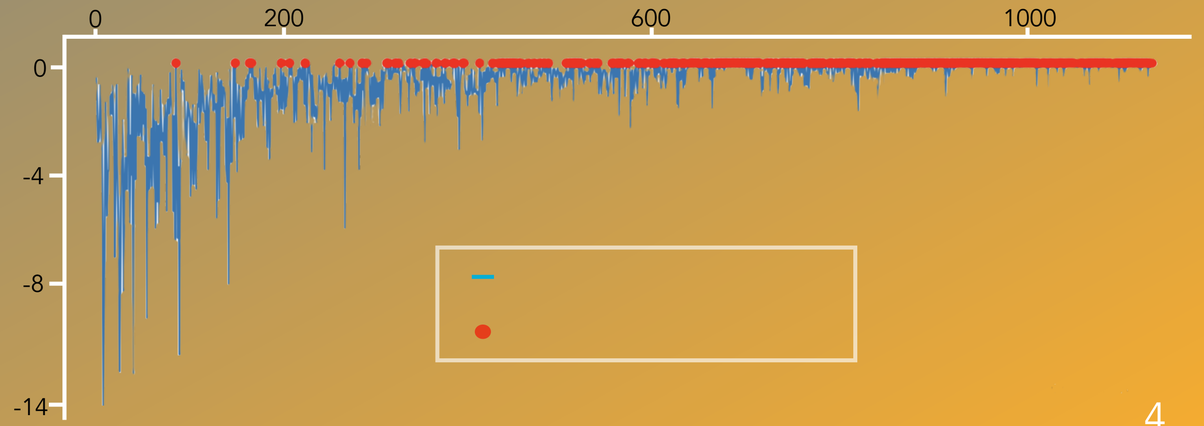
Gradient-based	Policy-gradient
$u_i \leftarrow u_i + \epsilon \frac{\partial J}{\partial u_i}$	$\pi_\theta \leftarrow \frac{\partial R}{\partial \theta_i}$



Grid World example

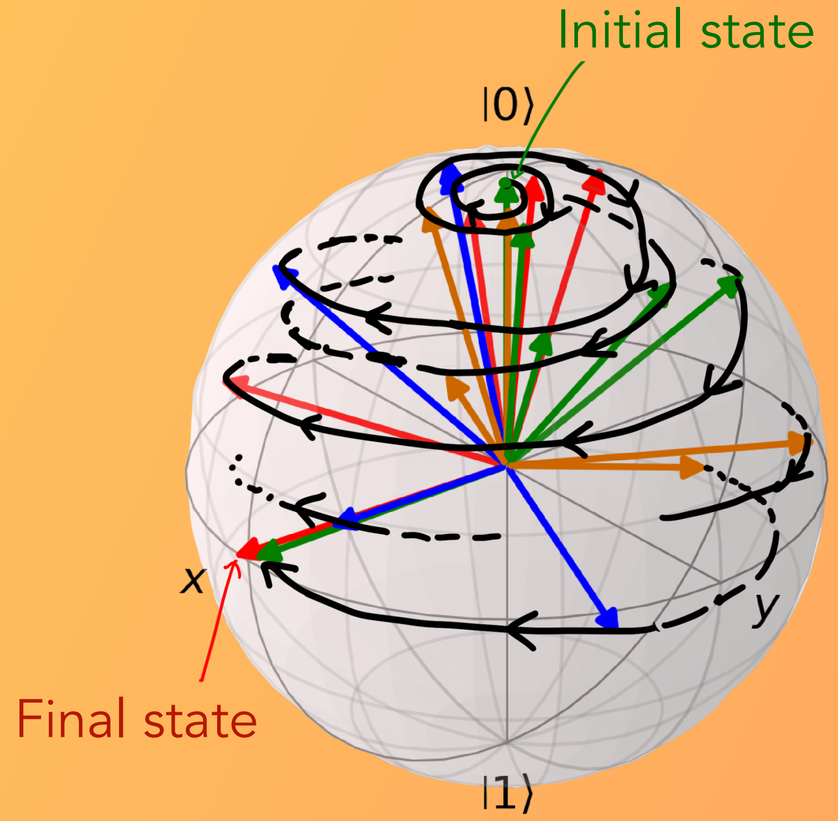
- Stable Baselines
  RL Policy
- Gymnasium
  step(), rest(), etc.
- Reward function
- Terminated & Truncated





Hadamard Transformation

- The RL algorithm learns to perform a Hadamard transformation, bringing the up state to the + state on the Bloch sphere with >99% fidelity.
- Training reduced the number of steps needed to achieve high fidelity, reaching the goal in 16 steps (visualized by 16 arrows/kets).
- After modifying the action space, the algorithm achieved the transformation in a single step.



RL Hadamard rotation

Integration with qutip-qoc

- Input:
  - Objective
  - control_parameters
  - tlist
  - algorithm_kwargs
- Output (Result class):
  - total_seconds
  - final infidelity
  - final_parameters

Addition of functionalities

- EarlyStopTraining callback fx:
  - Maximum episodes
  - Target fidelity achieved
  - Shorter pulse optimization
- Different control functions

RL code for qutip-qoc: https://github.com/qutip/qutip-qoc/blob/138e3d854c0a76abcea5b66602ff6b6671571931/src/qutip_qoc/_rl.py

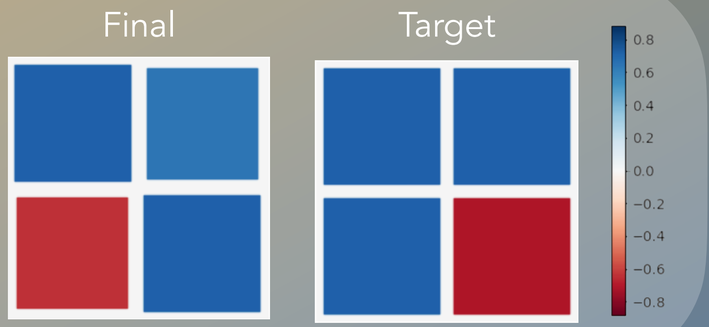
Hadamard No noise

Infidelity: 0.002173…

Final parameters: […]

Time: 0.399s

Message: Stop training because an episode with infidelity <= target infidelity was found



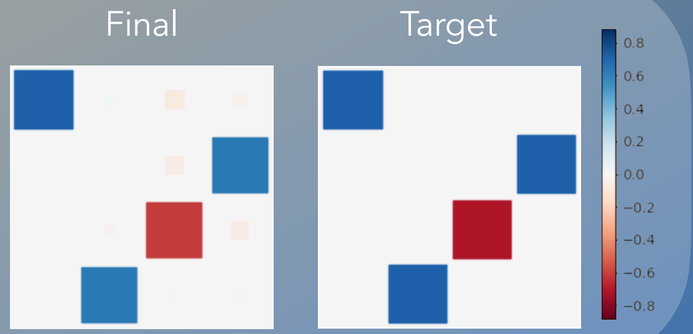
Hadamard noise

Infidelity: 0.008910…

Final parameters: [#19 - #(k*10)]

Time: 7s - 20s

Message: Stop training because an episode with infidelity <= target infidelity was found



GRAPE

Infidelity: 0.0439…

Time: 1.357s

Final parameters: #100x2

GOAT

Infidelity: 0.0451…

Time: 0.267s

Final parameters: #3x2

JOPT

Infidelity: 0.0087…

Time: 1.915s

Final parameters: #3x2

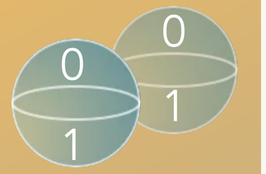
CRAB

Infidelity: 0.0456…

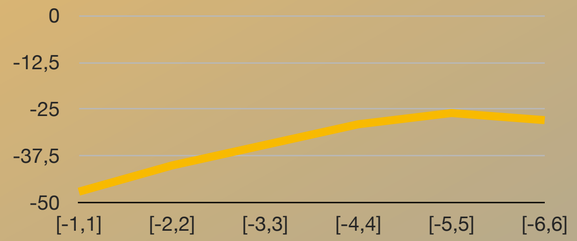
Time: 1.849s

n_params: 3,6,9…

CNOT No noise

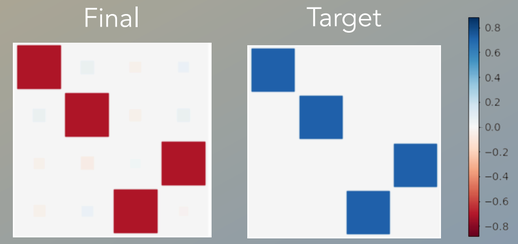


tlist = (0, 2π, 100) , max iter = 3000, bounds = [-1, 1] ✗ ? ✓ Also extending training (max_iter=5000)

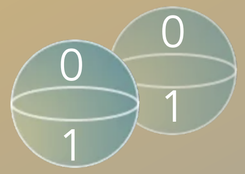


bounds = [-5, 5] (Infidelity ~ 5%) mean_rew ~ -26

bounds = [-6, 6] Always ✓ More than 1 min. →

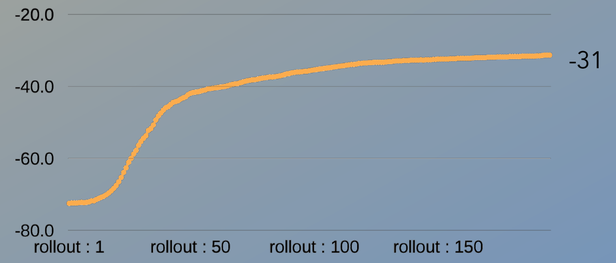


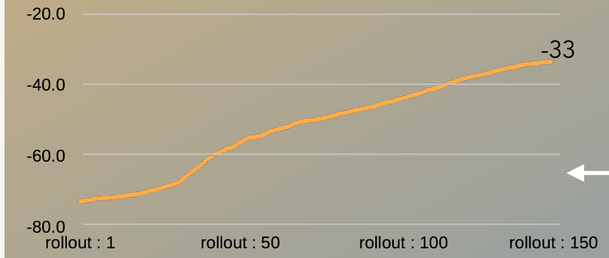
CNOT Noise



tlist = (0, 2π, 100) , max iter = 3000, bounds = [-6, 6]

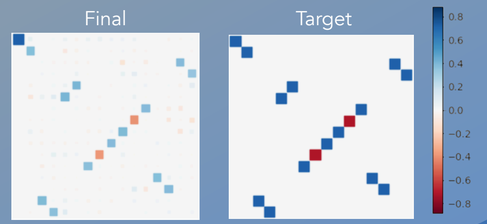
max iter = 4000 →





← bounds = [-10, 10] Also with max iter = 4000 (plateau -31)

It takes several minutes



Adding σ_z , episode duration π,
bounds = [-25, 25] —> final infid. down to 7%

	GRAPE		GOAT		JOPT		CRAB	
(*)	Infid: 0.11	Time: 3.02s	Infid: 0.36	Time: 3.30s	Infid: 0.36	Time: 3.80s	Infid: 0.46	Time: 13.23s
(**)	Infid: 0.11	Time: 27.65s	Infid: 0.27	Time: 2.84s	Infid: 0.27	Time: 2.64s	Infid: 0.18	Time: 9.55s

Possible future developments

- Different control **functions** (sin, gaussian, etc.)
- Experimenting with **alternative policies** for the RL
- **Multiple objectives:** optimum field and minimizing the Fluence (energy used)
- **Fine-tuning**: pre-training on quantum control tasks to reduce RL training time
- **Hybrid approach**: feedback from real-world systems

Thanks for your attention

Other tests for CNOT with noise. Using 3x(2) control H

tlist = (0, 2π, 100) , max iter = 3000, bounds = [-6, 6] → ep_rew_mean = -25, plateau

tlist = (0, 2π, 100) , max iter = 3000, bounds = [-10, 10] → ep_rew_mean = -29,3

---

tlist = (0, π, 100) , max iter = 3000, bounds = [-6, 6] → ep_rew_mean = -25 plateau, final Infid. 60%

tlist = (0, π, 100) , max iter = 3000, bounds = [-10, 10] → ep_rew_mean = -25 plateau, final Infid. 60%

tlist = (0, π, 100) , max iter = 3000, bounds = [-13, 13] → ep_rew_mean = -25 plateau, final Infid. 40%

tlist = (0, π, 100) , max iter = 3000, bounds = [-16, 16] → ep_rew_mean = -24 plateau, final Infid. 20%

tlist = (0, π, 100) , max iter = 3500, bounds = [-19, 19] → ep_rew_mean = -24, final Infid. 14%

tlist = (0, π, 100) , max iter = 5000, bounds = [-25, 25] → ep_rew_mean = -27 plateau, final Infid. 7% (12 min)

Other tests for Hadamard and CNOT

Reward function = Fidelity - step_penalty_term → Similar results as before…

$0 < f < 1$

1 —> 0.2