Deep reinforcement learning to uncover autonomous navigation strategies in turbulent flows

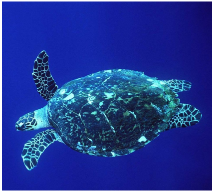
Aurore Loisy, with Christophe Eloy

Aix-Marseille Univ, CNRS, Centrale Marseille, IRPHE, Marseille, France

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 834238).

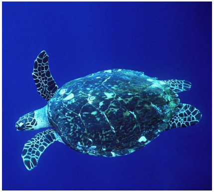
Deciphering animal navigation

All livings organisms have evolved autonomous navigation strategies to survive, e.g., to travel long distances efficiently or to locate food, shelter, and mates.

Deciphering animal navigation

All livings organisms have evolved autonomous navigation strategies to survive, e.g., to travel long distances efficiently or to locate food, shelter, and mates.

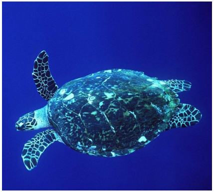



Instantaneous behaviour described by: motor action $= \pi$(sensory cues).

What should this function be to allow the animal to ultimately reach its target?

Deciphering animal navigation

All livings organisms have evolved autonomous navigation strategies to survive, e.g., to travel long distances efficiently or to locate food, shelter, and mates.

Instantaneous behaviour described by: motor action $= \pi$(sensory cues).

What should this function be to allow the animal to ultimately reach its target?

Hypothesis: techniques from artificial intelligence (deep reinforcement learning) can help us to reverse-engineer navigation algorithms used by animals.

Two examples: vertical migration of plankton and olfactory search by insects.

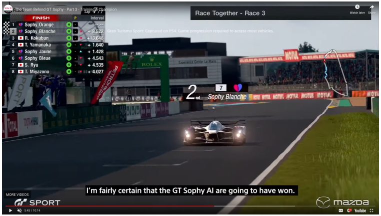
Deep Reinforcement Learning for games

DQN (Nature, 2015)

AlphaGo (Nature, 2016)

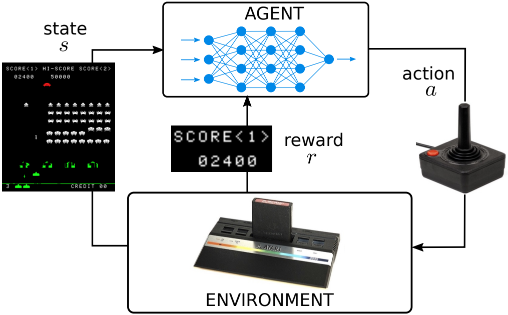
AlphaStar (Nature, 2019)

GT Sophy (Nature, 2022)



Reinforcement learning

How should an agent behave in order to maximize a long-term objective



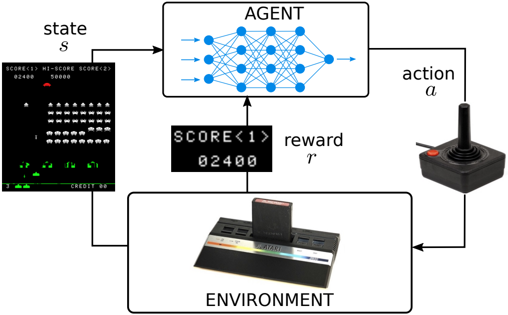
The agent's behavior is controlled by the policy π.

The policy maps each state to an action: $a = \pi(s)$.

In deep RL, the policy is a deep neural network (universal approximator).

Reinforcement learning

How should an agent behave in order to maximize a long-term objective



The agent's behavior is controlled by the policy π.

The policy maps each state to an action: $a = \pi(s)$.

In deep RL, the policy is a deep neural network (universal approximator).

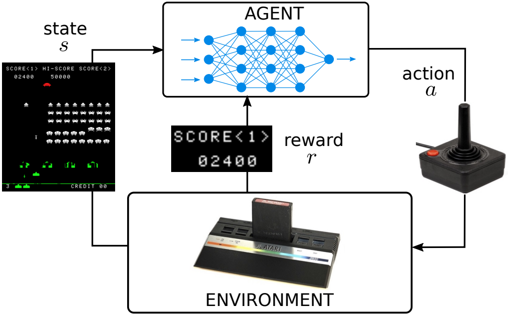
Each episode ("game") is a sequence $s_1, a_1, r_1, s_2, \ldots, a_T, r_T$.

The return ("final score") for an episode is $G = \sum_{t=1}^{T} r_t$.

We seek the optimal policy π^*, defined as $\pi^* = \arg\max_\pi \mathbb{E}_\pi[G]$.

Reinforcement learning

How should an agent behave in order to maximize a long-term objective



The agent's behavior is controlled by the policy π.

The policy maps each state to an action: $a = \pi(s)$.

In deep RL, the policy is a deep neural network (universal approximator).

Each episode ("game") is a sequence $s_1, a_1, r_1, s_2, \ldots, a_T, r_T$.

The return ("final score") for an episode is $G = \sum_{t=1}^{T} r_t$.

We seek the optimal policy π^*, defined as $\pi^* = \arg\max_\pi \mathbb{E}_\pi[G]$.

RL provides the methods to compute or approximate the optimal policy.

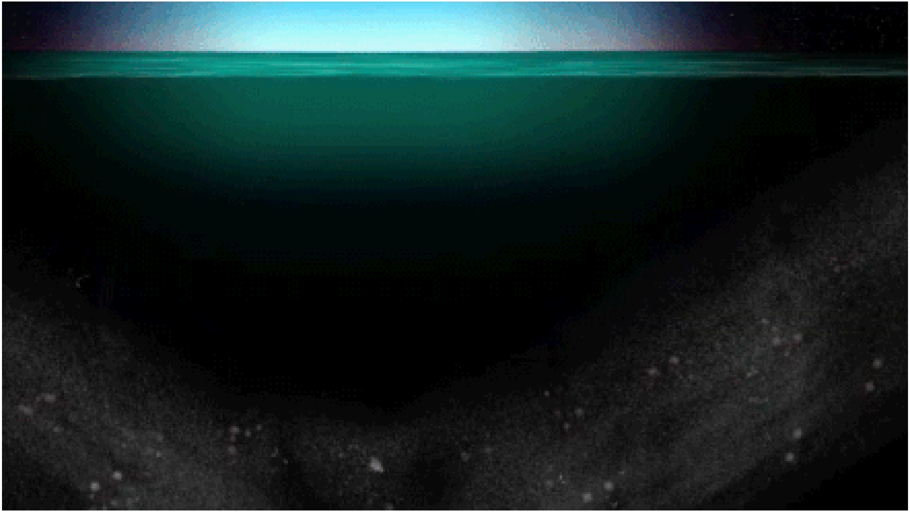
These methods are based on the agent interacting with the environment: the agent learns by doing (self-generated experience is our "training set").

Vertical migration of plankton

with

Rémi Monthiller (PhD 2022) and Selim Mecanna (PhD student)

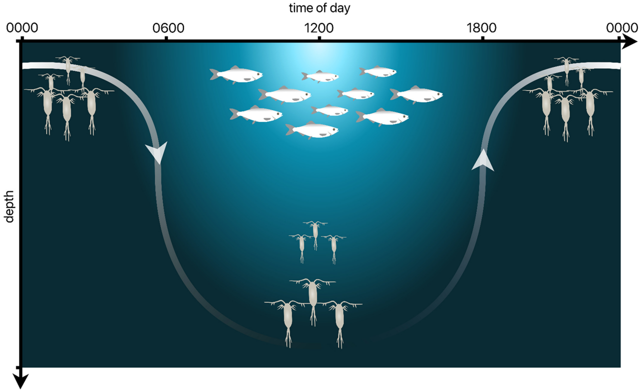
Diel vertical migration of plankton

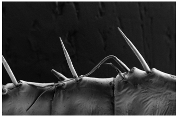


Credit: Woods Hole Oceanographic Institution

Every day, millimetric zooplanktons travel hundreds of meters in the water column (equivalent to 10 daily marathons for humans).

Diel vertical migration of plankton





Planktons are advected by the flow.

But many can swim and sense fluid motion relative to their bodies.

Can hydrodynamic signals be exploited to migrate more efficiently?

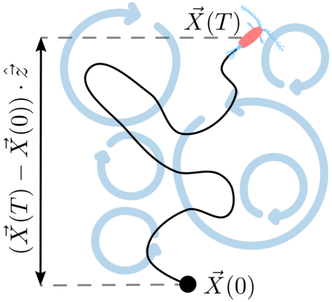
Model problem

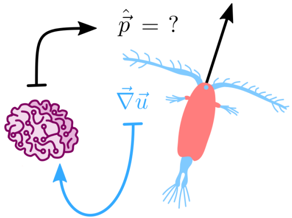
Equation of motion: $\dot{\boldsymbol{X}} = \boldsymbol{u}(\boldsymbol{X}, t) + V\hat{\boldsymbol{p}}$

Sensors: local velocity gradient $\boldsymbol{\nabla u}$ (and $\hat{\boldsymbol{z}}$)

Control: direction of motion $\hat{\boldsymbol{p}}$

Objective: maximize vertical distance travelled





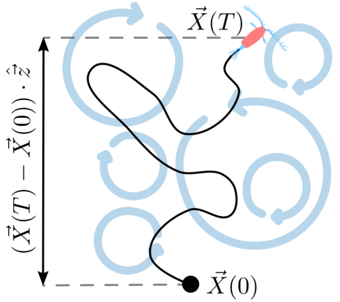
Model problem

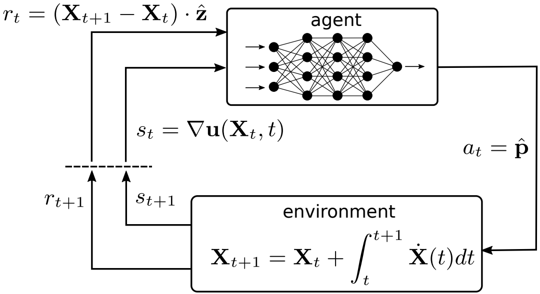
Equation of motion: $\dot{\boldsymbol{X}} = \boldsymbol{u}(\boldsymbol{X}, t) + V\hat{\boldsymbol{p}}$

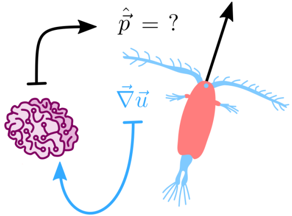
Sensors: local velocity gradient $\boldsymbol{\nabla u}$ (and $\hat{\boldsymbol{z}}$)

Control: direction of motion $\hat{\boldsymbol{p}}$

Objective: maximize vertical distance travelled

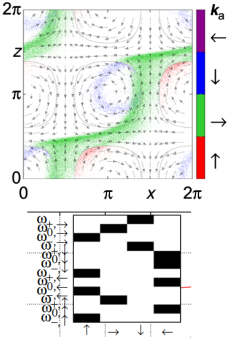






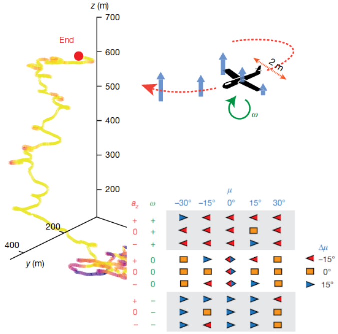
Past studies all used discrete states and actions with 'classical' RL (not deep)

(up to a few exceptions)



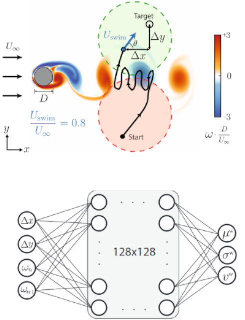
vertical migration in TGV

Colabrese, Gustavsson, Celani & Biferale (Phys. Rev. Lett., 2017)



ascending thermal plumes

Reddy, Wong-Ng, Celani, Sejnowski & Vergassola (Nature, 2018)

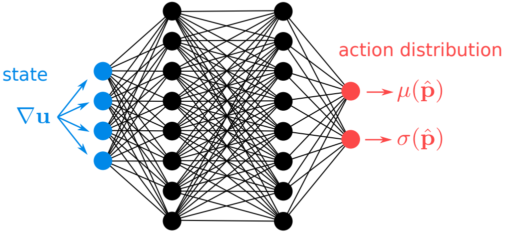


point-to-point navigation

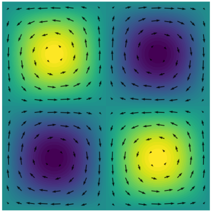
Gunnarson, Mandralis, Novati, Koumoutsakos & Dabiri (Nat. Commun., 2021)

Here, we use **continuous states and actions** with a **deep-NN policy**

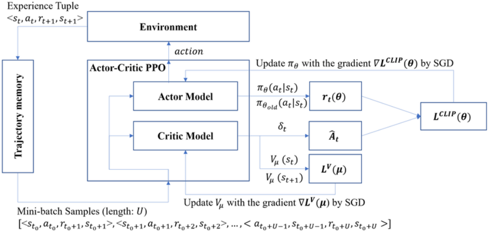
agent: stochastic policy



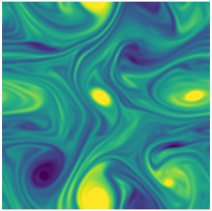
environment 1: Taylor-Green



RL: actor-critic PPO



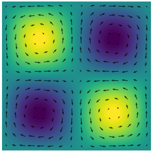
environment 2: 2D turbulence



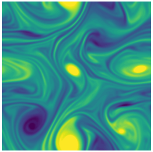
pseudo-spectral DNS (256x256)
with large-scale stochastic forcing

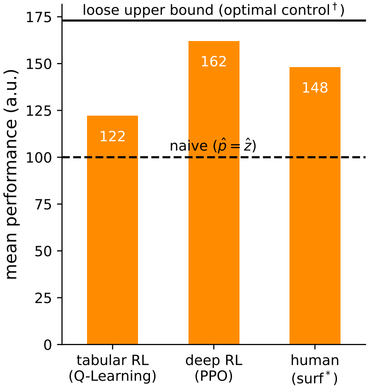
Benchmarking (preliminary)

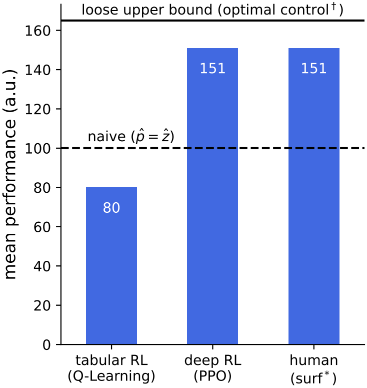
Taylor-Green



2D turbulence



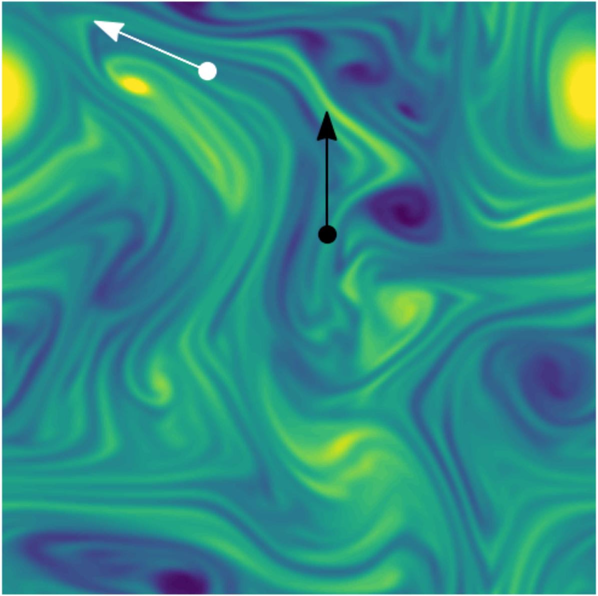




* Monthiller, Loisy, Koehl, Favier & Eloy (Phys. Rev. Lett, 2022)

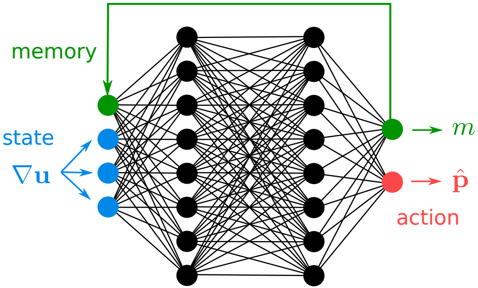
† Calascibetta, Biferale, Borra, Celani & Cencini (arXiv.2305.04677, 2023)

Example trajectory (preliminary)



Conclusions and perspectives

- Conclusions
  - information provided by local velocity gradients can be exploited to travel effectively much faster in complex flows → biology implications?
  - deep RL provides better, more robust solutions than 'vanilla' Q-learning and deep nets remain fairly cheap to train (half day on Intel Core i7-10700)
- Perspectives
  - physical interpretation of the NN policy
  - 3D turbulence
  - agent with memory (recurrent NN)



· challenging to train (very stochastic environment → huge variance)

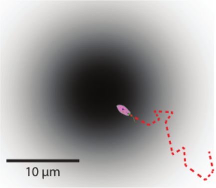
· expected gain is small (memoryless PPO is already close to OC)

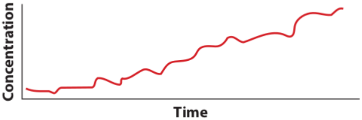
Olfactory search by insects

Search strategies vs scales

microscopic scale (e.g., bacteria)

molecular diffusion



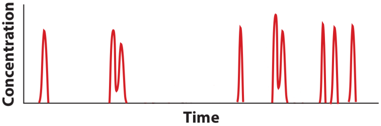


strategy: **chemotaxis**

directional motion in response to a chemical signal (gradient ascent)

macroscopic scale (e.g., insects)

turbulent dispersion

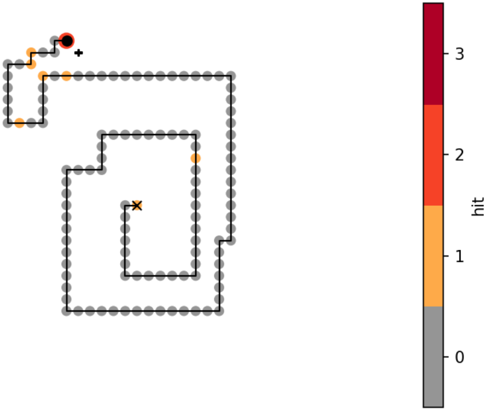


strategy?

illustrations from Reddy, Murthy & Vergassola (Annu. Rev. Condens. Matter Phys., 2022)

Olfactory search POMDP

formulated by Vergassola, Villermaux & Shraiman (Nature, 2007)



What is the best strategy to find the source as quickly as possible?

Physical model of dispersion and detection

formulated by Vergassola, Villermaux & Shraiman (Nature, 2007)

- **Source**
  - point source at a fixed location
  - odor particles emitted at a rate R with finite lifetime τ
  - isotropic medium characterized by an effective diffusivity D
  - characteristic lengthscale of dispersion $\lambda = \sqrt{D\tau}$

- **Agent**
  - sphere of radius a
  - takes a "sniff" during a time Δt (absorbs odor particles diffusing to its surface)
  - then moves by one body length $\Delta x = 2a$

- **Independent physical parameters**: R, λ, Δt, Δx

- **Dimensionless parameters**
  - dimensionless dispersion lengthscale $\mathcal{L} = \dfrac{\lambda}{\Delta x}$
  - dimensionless source intensity $\mathcal{I} = R\Delta t$

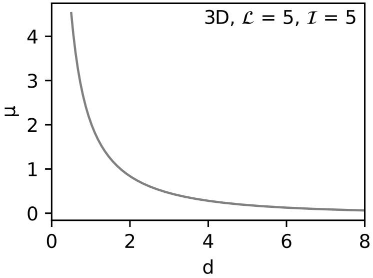
Observation model

formulated by Vergassola, Villermaux & Shraiman (Nature, 2007)

- The **mean** number of hits μ **decays with the distance** d to the source

$$\mu(d) = \frac{\mathcal{I}}{2d} \exp\left(\frac{-d}{\mathcal{L}}\right) \qquad \text{(in 3D)}$$

where $\mathcal{L}$ is the dispersion lengthscale and $\mathcal{I}$ is the source intensity.

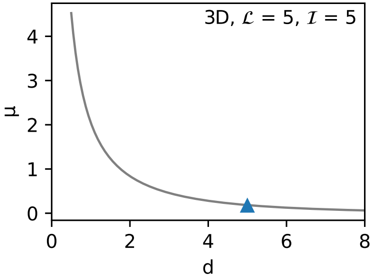


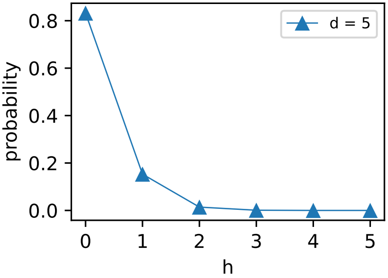
Observation model

formulated by Vergassola, Villermaux & Shraiman (Nature, 2007)

- The **actual** number of hits h is **drawn from a Poisson distribution** with mean μ:

$$\Pr(h; \mu) = \frac{\mu^h e^{-\mu}}{h!} \quad \text{for } h = 0, 1, 2, \ldots$$



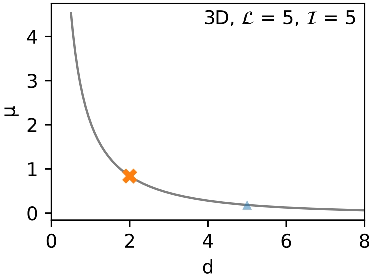


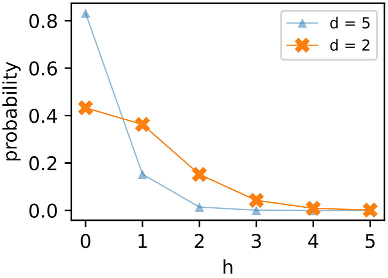
Observation model

formulated by Vergassola, Villermaux & Shraiman (Nature, 2007)

- The **actual** number of hits h is **drawn from a Poisson distribution** with mean μ:

$$\Pr(h;\mu) = \frac{\mu^h e^{-\mu}}{h!} \quad \text{for } h = 0, 1, 2, \ldots$$

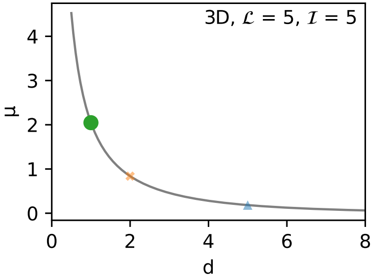


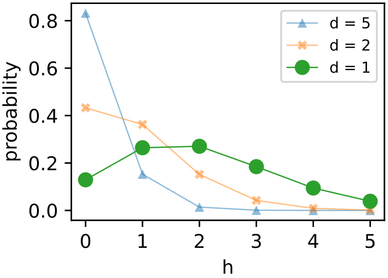


formulated by Vergassola, Villermaux & Shraiman (Nature, 2007)

- The **actual** number of hits h is **drawn from a Poisson distribution** with mean μ:

$$\Pr(h;\mu) = \frac{\mu^h e^{-\mu}}{h!} \quad \text{for } h = 0, 1, 2, \ldots$$



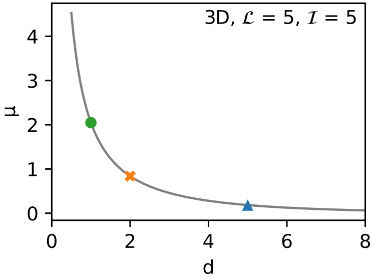


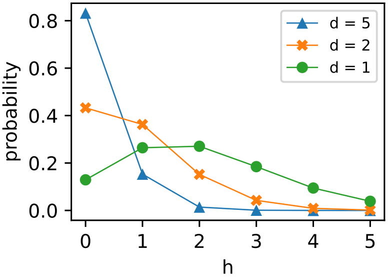
Observation model

formulated by Vergassola, Villermaux & Shraiman (Nature, 2007)

- The **actual** number of hits h is **drawn from a Poisson distribution** with mean μ:

$$\Pr(h;\mu) = \frac{\mu^h e^{-\mu}}{h!} \quad \text{for } h = 0, 1, 2, \ldots$$

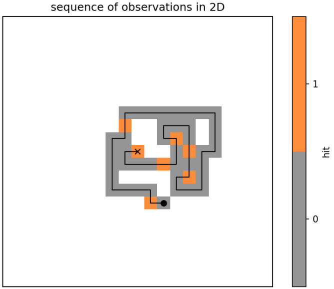


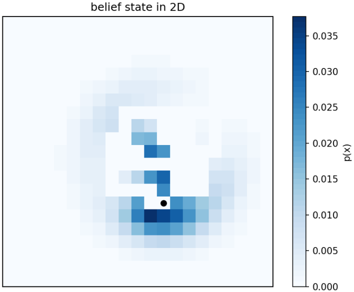


Hits provide noisy information about the distance to the source

Belief-MDP formulation

The agent remembers the entire sequence of past actions and observations.
All past information can be encoded in the agent's **belief state** $s = [\boldsymbol{x}^a, \Pr(\boldsymbol{x})]$.





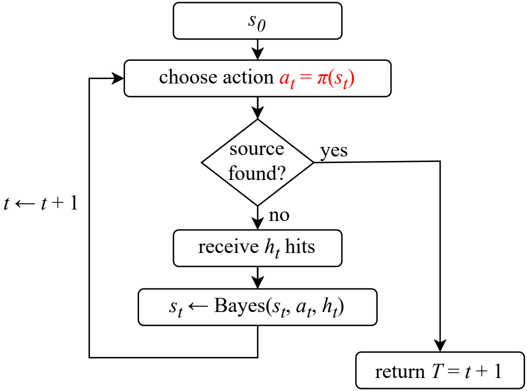
After observing h hits in $\boldsymbol{x}^a$, $\Pr(\boldsymbol{x})$ is **updated using Bayes' rule**:

$$\Pr(\boldsymbol{x}|\boldsymbol{x}^a, h) \;\propto\; \Pr(h|\boldsymbol{x}, \boldsymbol{x}^a)\; \Pr(\boldsymbol{x})$$

posterior likelihood prior

(given by the detection model)

Optimal policy



The agent's behavior is controlled by the policy π.

The policy maps each belief state to an action: $a = \pi(s)$.

The **performance** of a policy π is measured by $\mathbb{E}_\pi[T]$, the **expected number of steps to reach the source** when acting according to π.

We seek the **optimal policy** π^*, defined as $\pi^* = \arg\min_\pi \mathbb{E}_\pi[T]$.

Our problem is a **belief-MDP**: an MDP where states are replaced by belief states.

Optimal value function

The **optimal value function** $v^*(s)$ of a belief state s is defined as the minimum, over all policies, of the expected number of steps remaining to find the source when starting from belief state s:

$$v^*(s) = \min_{\pi} v^{\pi}(s) \qquad \text{where} \quad v^{\pi}(s) = \mathbb{E}_{\pi}[T - t | s_t = s].$$

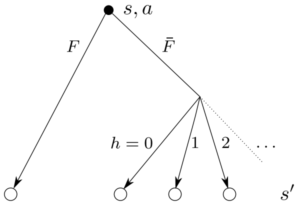
Optimal value function

The **optimal value function** $v^*(s)$ of a belief state s is defined as the minimum, over all policies, of the expected number of steps remaining to find the source when starting from belief state s:

$$v^*(s) = \min_\pi v^\pi(s) \qquad \text{where} \quad v^\pi(s) = \mathbb{E}_\pi[T - t | s_t = s].$$

Given $v^*(s)$, the **optimal policy** consists in choosing the action that minimizes the expected number of remaining steps $v^*(s')$:

$$\pi^*(s) = \arg\min_a \sum_{s'} \Pr(s'|s,a) v^*(s').$$



Here $\Pr(s'|s,a)$ is the probability of transitioning from belief state s to next belief state s' after executing action a.

Solving Bellman optimality equation

The optimal value function satisfies the **Bellman optimality equation**

$$v^*(s) = 1 + \min_a \sum_{s'} \Pr(s'|s,a) v^*(s') \qquad \text{for all nonterminal belief states } s.$$

Finding the **optimal policy** π^* amounts to finding v^* that satisfies this equation.

Curse of history: number of belief states grows as $(N_{\text{actions}} \times N_{\text{observations}})^{N_{\text{steps}}}$.
$\rightarrow$ The optimal value function **cannot be computed exactly**.

Solving Bellman optimality equation

The optimal value function satisfies the **Bellman optimality equation**

$$v^*(s) = 1 + \min_a \sum_{s'} \Pr(s'|s,a) v^*(s') \qquad \text{for all nonterminal belief states } s.$$

Finding the **optimal policy** π^* amounts to finding v^* that satisfies this equation.

Curse of history: number of belief states grows as $(N_{\text{actions}} \times N_{\text{observations}})^{N_{\text{steps}}}$.

$\rightarrow$ The optimal value function **cannot be computed exactly**.

$\rightarrow$ We seek **approximate solutions**.

Standard approach

Value iteration

$$v^*(s) \leftarrow 1 + \min_a \sum_{s'} \Pr(s'|s,a) v^*(s') \qquad \text{on a } \textbf{subset} \text{ of belief states}$$

will converge to the fixed point $v = v^*$.

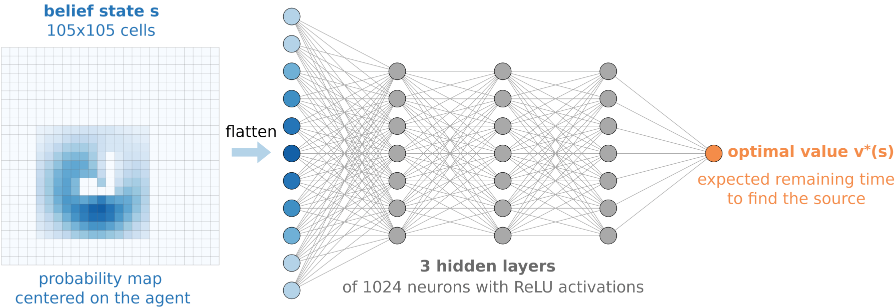
Different solvers use different heuristics to construct the subset of belief states.

Heinonen, Biferale, Celani & Vergassola (Phys. Rev. E, 2023)

Using model-based DRL

We **approximate** $v^*(s)$ by a **neural network** $v^*(s; \boldsymbol{w}^*)$. The optimal weights $\boldsymbol{w}^*$ **minimize** the **residual error** on the Bellman optimality equation (the "loss")

$$v^*(s; \boldsymbol{w}^*) \approx 1 + \min_a \sum_{s'} \Pr(s'|s, a) v^*(s'; \boldsymbol{w}^*) \qquad \text{for } s \sim \hat{\pi}^* \text{ derived from } v^*$$



The NN is trained using a custom **deep reinforcement learning** algorithm.

Loisy & Eloy (Proc. R. Soc. A, 2022)

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s, a, h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s,\ a,\ h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s,\, a,\, h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s, a)[1 + v(s'; w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s; w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s, a)[1 + v(s'; w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s,\ a,\ h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s, a, h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s, a, h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s, a)[1 + v(s'; w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s; w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s, a)[1 + v(s'; w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s, a, h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \mathrm{Bayes}(s,\ a,\ h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \mathrm{Bayes}(s, a, h)$

end if

Reinforcement learning (a.k.a. learning from experience)

Initialize value function v with random weights w

Initialize belief state s

Loop forever

Compute all s' accessible from s for every action a

Compute targets $y = \min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

Gradient descent step: adjust w to reduce loss $L(w) = (y - v(s;w))^2$

Select action from current policy: $a = \arg\min_a \sum_{s'} \Pr(s'|s,a)[1 + v(s';w)]$

if *source found* **then**

Reinitialize s for a new episode

else

Receive a hit h

Transition to a new belief state: $s \leftarrow \text{Bayes}(s, a, h)$

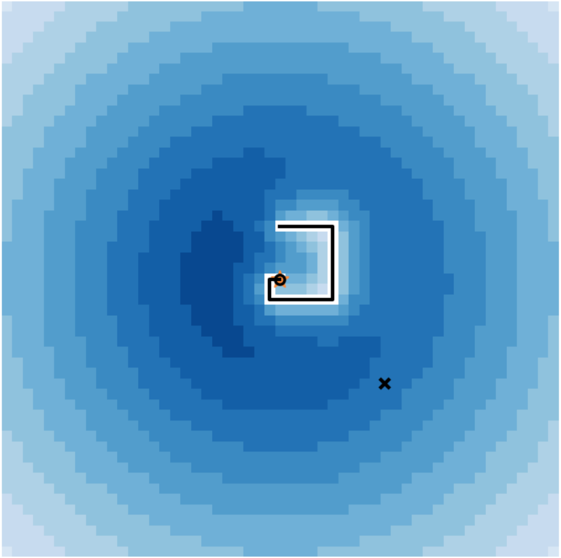
end if

Reinforcement learning (a.k.a. learning from experience)

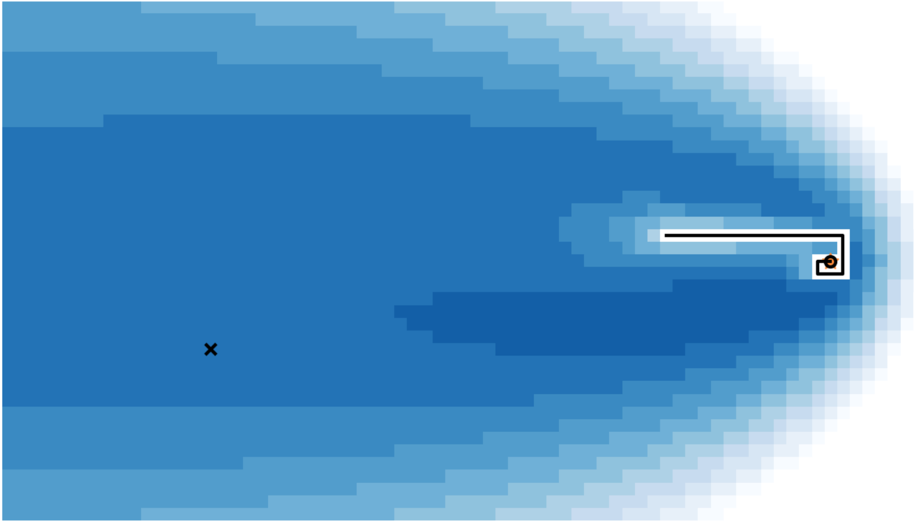
```
Initialize replay memory to capacity memory_size
Initialize value function v with random weights w
Initialize target value function v⁻ with random weights w⁻ = w
converged_weights ← False
it ← 0
while not converged_weights do
                                                    // Generate new experience
    epsilon ← max(epsilon_init ∗ exp(−it/epsilon_decay), epsilon_floor)          // decaying ϵ of ϵ-greedy
    m ← 0
    episode_complete ← True
    while m < new_transitions_per_it do
        if episode_complete then
            initialize belief state s for a new episode
            episode_complete ← False
        end if
        s ← apply_random_symmetry(s)                         // randomize over symmetries of the problem
        for all actions a, compute all s′ accessible from s (i.e. all outcomes (found/not found and hits))
        store (s, a, s′) in replay memory
        m ← m + 1
        with probability epsilon select a random action a,                      // ϵ-greedy exploration
        otherwise select action a = arg min_a Σ_s′ Pr(s′|s, a)[1 + v(s′; w)]          // according to current policy
        s ← make_step_in_env (s, a)                 // transition to a new belief state according to action
        episode_complete ← is_episode_complete(s)
    end while
                                        // Update weights by stochastic gradient descent
    for gd_step = 1, gd_steps_per_it do
        Sample minibatch_size transitions (s, a, s′) from replay memory
        For each transition, compute targets y = min_a Σ_s′ Pr(s′|s, a)[1 + v⁻(s′; w⁻)]        // using delayed target network
        Perform a gradient descent step on (y − v(s; w))² with respect to the network parameters w
    end for

    converged_weights ← are_weights_converged()
    it ← it + 1
    every update_target_network_it iterations, reset v⁻ = v
end while
```

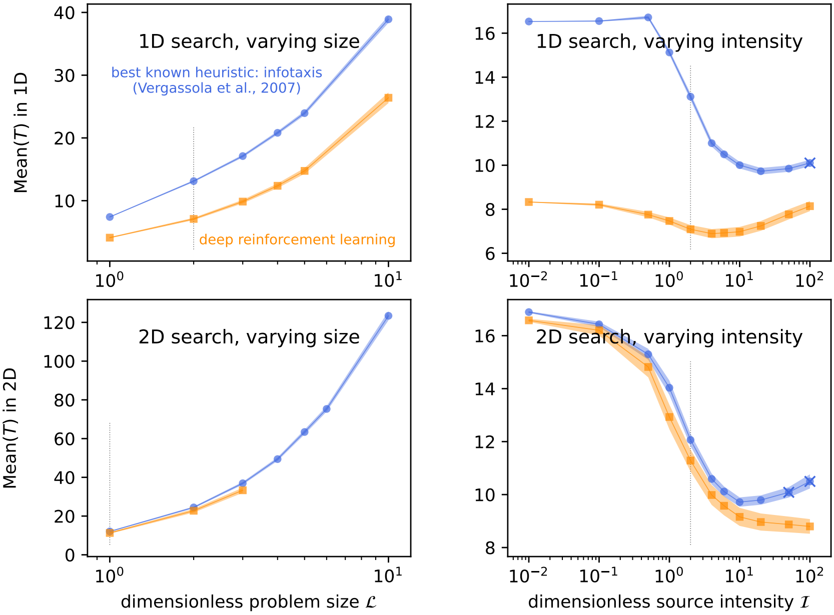
Example of quasi-optimal searches: no mean wind



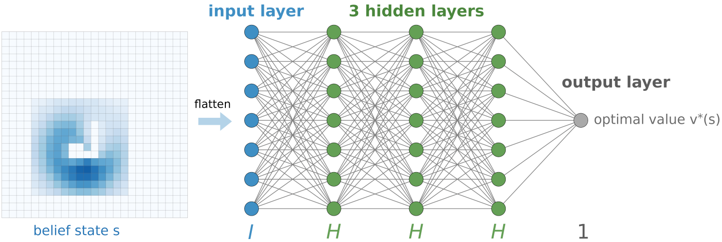
Example of quasi-optimal searches: directional mean wind



Results for various search conditions



Scaling up



Number of free parameters:

$$N = H(I + 2H + 4) + 1$$

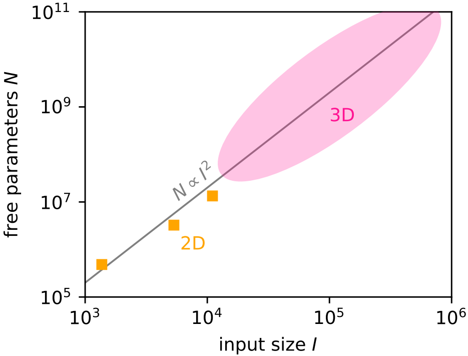
Neurons per hidden layer:

$$H \sim I$$

Free parameters vs input size:

$$\Rightarrow N \propto I^2$$

Scale-up: multiscale coarse-graining?



Benchmarking

Two concurrent methods to solve approximately the Bellman optimality equation:

$$v^*(s) = \mathcal{B}v^*(s) \qquad \text{with} \qquad \mathcal{B}v(s) = 1 + \min_a \sum_{s'} \Pr(s'|s,a)v(s')$$

point-based value iteration (PBVI)

approximate v^* by a piecewise-linear concave function (alpha-vectors) and repeatedly apply the Bellman operator $\mathcal{B}$ until convergence to its fixed point

deep reinforcement learning (DRL)

approximate v^* by a deep neural network and minimize the Bellman error by adjusting the network parameters using stochastic gradient descent

	PBVI	DRL
v^* representation	alpha-vectors	deep neural network
free parameters	$\sim 10^8$ parameters	$\sim 10^7$ parameters
optimality	✓ equally good policies	
solving speed	✓ 5 hours to converge	✗ 5 days to train
execution speed	✗ 2 days for 10^4 episodes	✓ 2 hours for 10^4 episodes

Loisy & Heinonen (Eur. Phys. J. E, 2023)

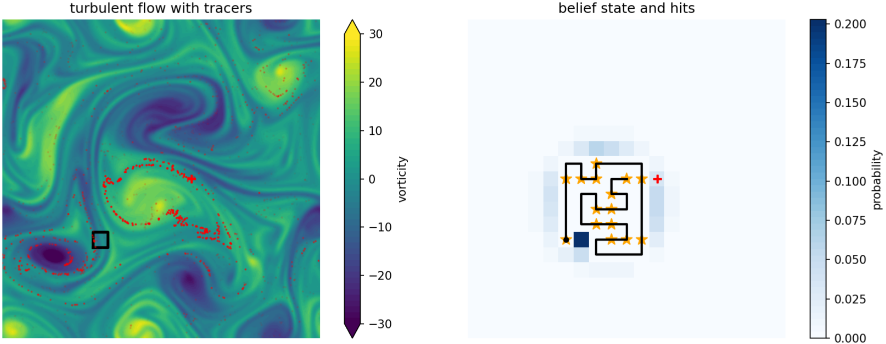
What about real turbulence?

How to find a source of odor in turbulence

What about real turbulence?

How to find a source of odor in ~~turbulence~~
a stochastic environment using uncorrelated observations

Agent trained in stochastic environment and tested in real 2D turbulence



Perspectives

- Can we really learn to navigate a turbulent odor plume?

		turbulent odor plume	
		synthetic data (no correlations)	real data (correlated detections)
memory	model-based (Bayesian map)	this talk	data-driven model
	model-free (variable to optimize)	recurrent neural network[†] finite-state controller[‡]	**next challenge for DRL?**

[†] Singh, van Breugel, Rao & Brunton (Nature Machine Intelligence, 2023)
[‡] Verano, Panizon & Celani (PNAS, 2023)

- Optimal strategies have applications in robotics.

- Are they also relevant to animal behaviour?