Inverse Q-Learning as a Tool to Investigate Behavior and its Neural Correlates

Albert-Ludwigs-Universität Freiburg

// BrainLinks
BrainTools

Joschka Boedecker jboedeck@cs.uni-freiburg.de
Hao Zhu hao.zhu.10015@gmail.com

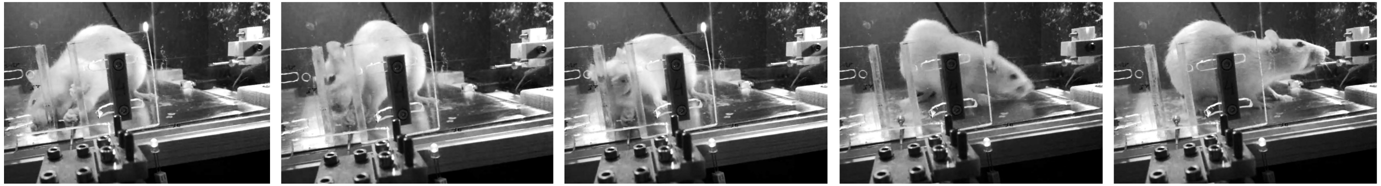
September 26th, 2023

UNI FREIBURG

Main Questions for Today

How do we explain goal-directed animal behavior given that we often see objectively non-optimal behavior? Which factors contribute? *What are the animals optimizing for?*

Response-Preparation Task (simplified):

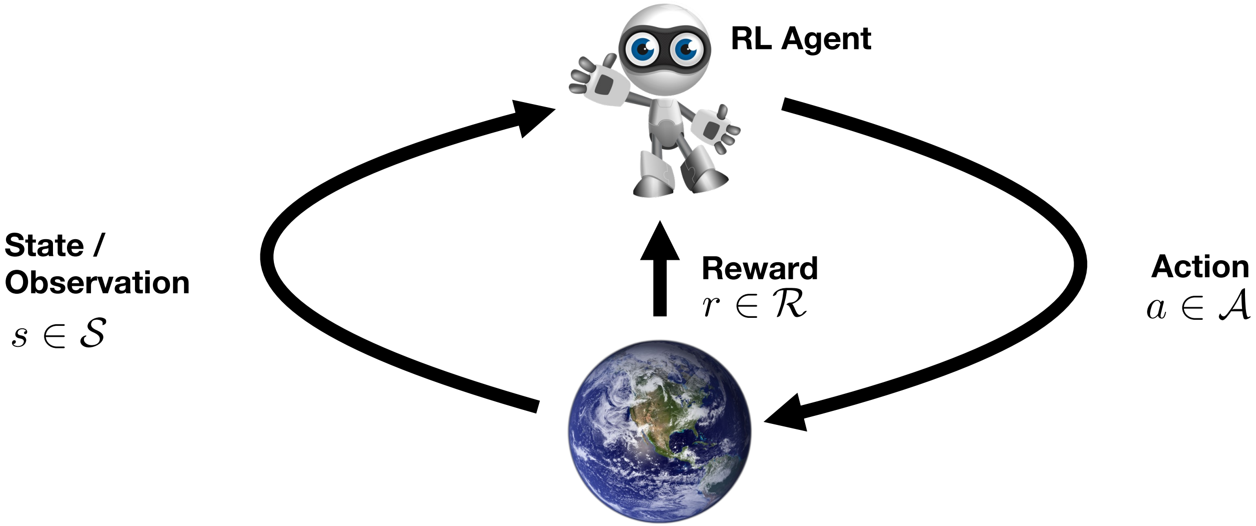


lever press vibration cue reward

- Press lever until cue (vibration) occurs (delay **1.6s**).
- After the cue, the rat has **0.6s** to release the lever
- If successful, the rat gets a treat.

Images courtesy of the Diester Lab

Reinforcement Learning in a Nutshell



$$p(s', r | s, a) = Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

Goal: find policy that maximizes expected long-term reward

States

3,1	3,2	3,3	3,4
2,1	2,2	2,3	2,4
1,1	1,2	1,3	1,4

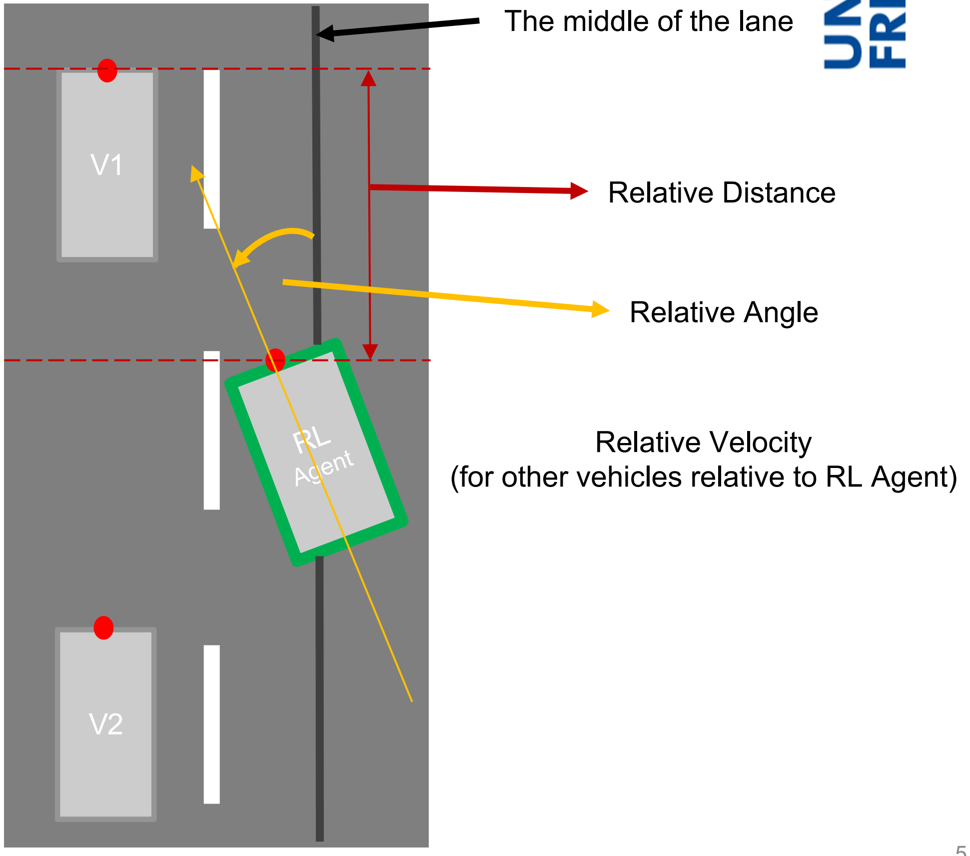
States in Autonomous Driving Application

20 features total:

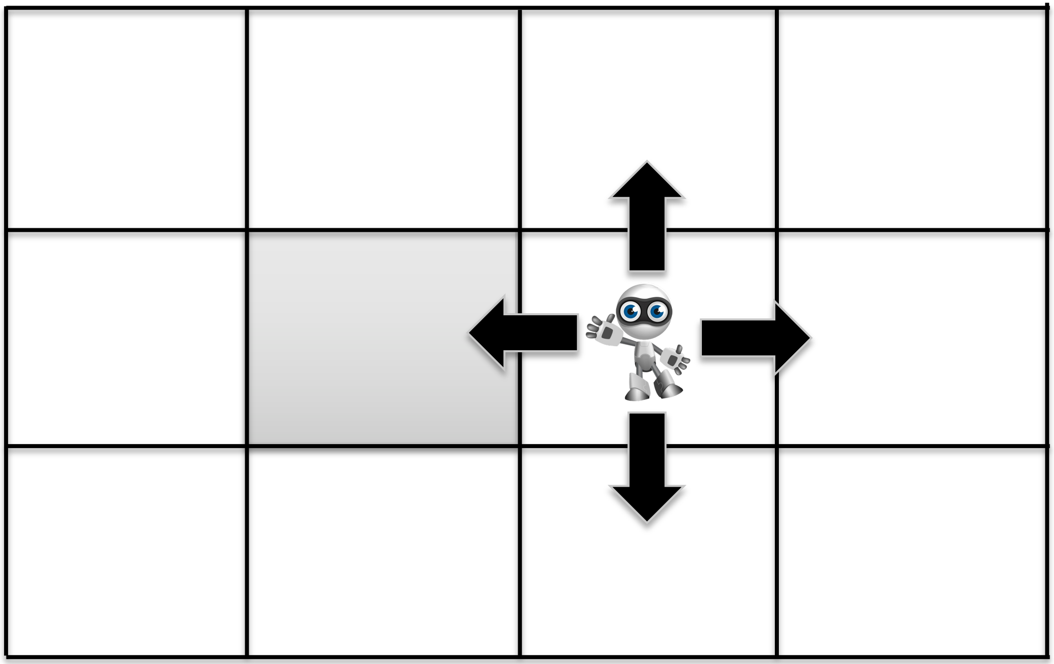
Max. 6 potential vehicles surrounding the RL agent

3 features per vehicle → 18
- Relative Distance
- Relative Velocity
- Relative Angle

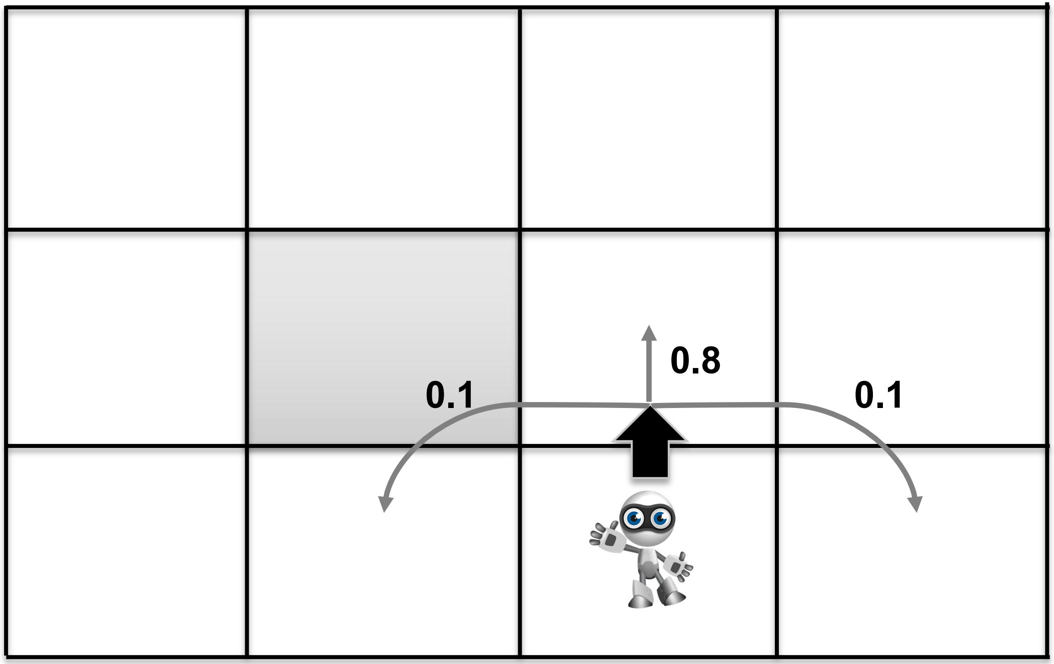
2 features describing the RL agent
- Velocity
- Relative Angle



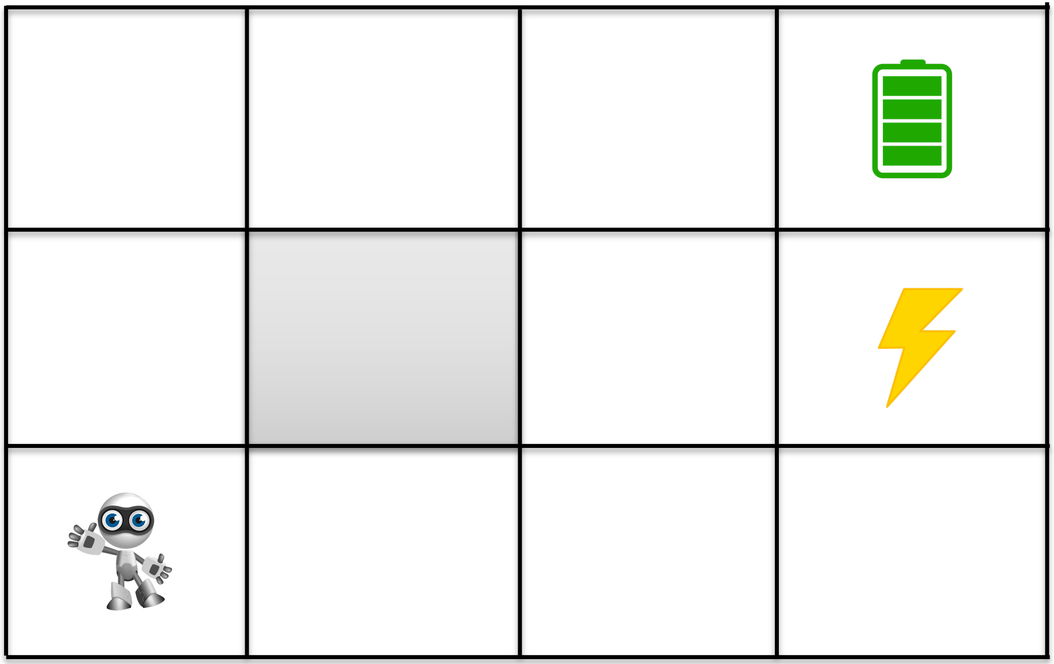
Actions



Transition Probabilities

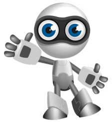


Rewards



- Stochastic vs deterministic
- Dense vs sparse (delayed)
- Magnitude

Rewards

-0.04	-0.04	-0.04	+1
-0.04		-0.04	-1
	-0.04	-0.04	-0.04

- Stochastic vs deterministic
- Dense vs sparse (delayed)
- Magnitude

Markov Decision Process

A finite Markov Decision Process (MDP) is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, p, \mathcal{R} \rangle$, where

- $\mathcal{S}$ is a finite number of states,
- $\mathcal{A}$ is a finite number of actions,
- p is the transition probability function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \mapsto [0,1]$
- $\mathcal{R}$ is a finite set of scalar rewards. We can then define expected reward

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

Markov Property:

$$\Pr\{S_{t+1}, R_{t+1} | S_t, A_t\} = \Pr\{S_{t+1}, R_{t+1} | S_t, A_t, \ldots, S_0, A_0\}$$

The future is independent of the past given the present.

Policy and overall Goal

Policy determines action selection for each state:

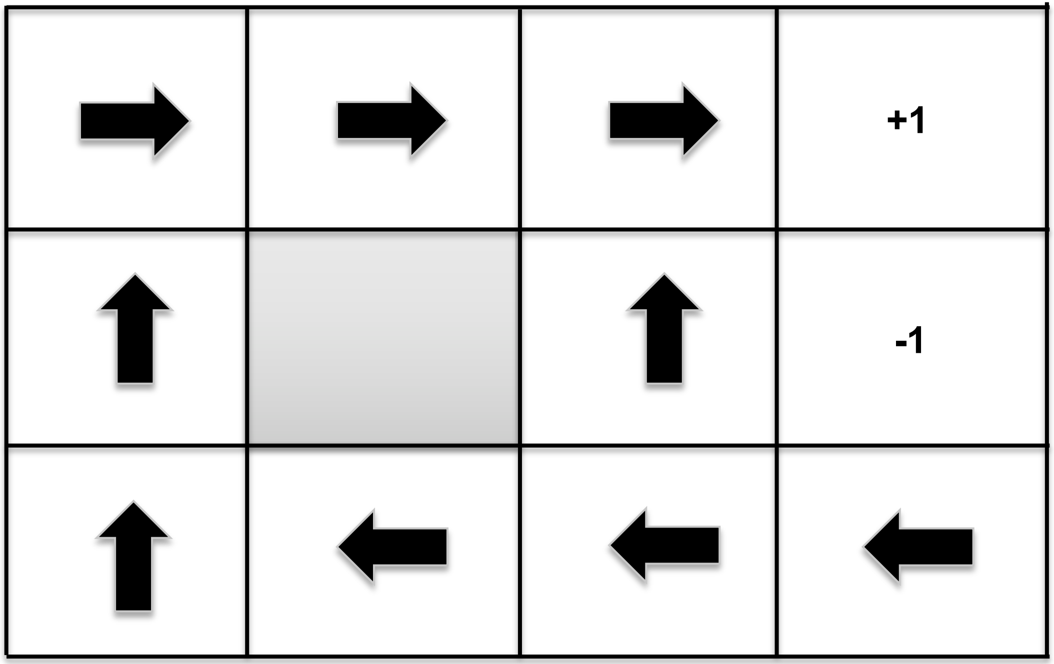
- Stochastic: $\pi(a|s) = \Pr[A_t = a | S_t = s]$
- Deterministic: $\pi(s) = a$

Goal for an RL agent in an MDP: find a policy that maximizes the expected return, i.e. the (discounted) cumulative reward: G_t

- Finite horizon: $\arg\max_\pi \mathbb{E}[\underbrace{R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T}_{G_t}]$
- Infinite horizon: $\arg\max_\pi \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}]$

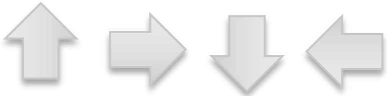
With discount $\gamma \in [0, 1]$ preventing infinite returns (converging geometric series)

Policy in MDP example



[Russel & Norvig, 2009]

Actions:



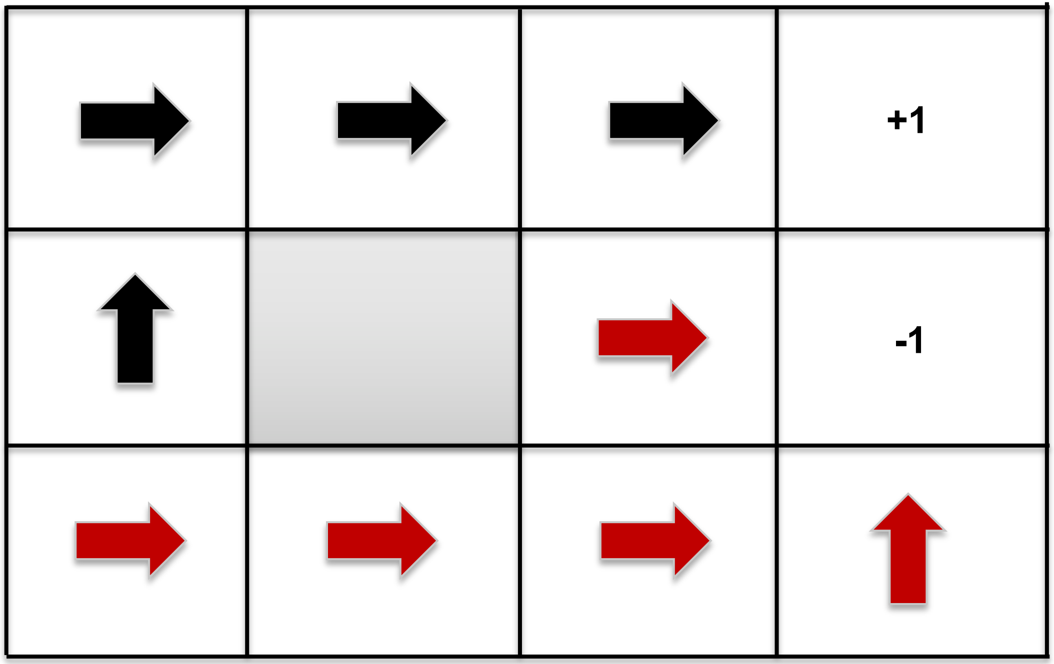
Probability of executing action successfully: 0.8

Rewards:
-0.04 / step

Question

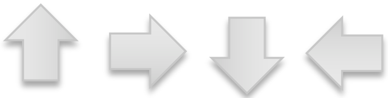
How will the policy change if we change the immediate reward to -2 instead of -0.04?

Policy in MDP example (changed rewards)



[Russel & Norvig, 2009]

Actions:



Probability of executing action successfully: 0.8

Rewards:
-2 / step

Value Function and Action-Value Function

Value Function $v_\pi(s)$ is the expected return when starting in s and following π :

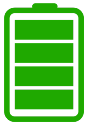
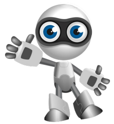
$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

Action-Value Function q_π is the expected return when starting in s, taking action a and following π thereafter:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

Value Function Example

$v_\pi(s)$ for immediate reward of -0.04, discount of 1:

0.812	**0.868**	**0.918**	
0.762		**0.660**	
	**0.655**	**0.611**	**0.388**

Bellman Optimality Equation

UNI FREIBURG

- A **Bellman Equation** expresses a relationship between the value of a state and the values of its successor states
- The **Bellman Optimality Equation** expresses that the value of a state under the optimal policy π_* must equal the expected return for the best action in that state

$$\begin{aligned} v_*(s) &= \max_a q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')] \end{aligned}$$

- The value function v_* is the **unique solution** to the Bellman Optimality Equation

Bellman Equation for v_*

$$v_*(s) = \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')]$$

We no longer need to search over all policies, only over all actions recursively!

Value Iteration Algorithm

An algorithm that turns the **Bellman Equation** into an **iterative update** to solve a given MDP

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$

From: [Sutton & Barto, 2018]

Bellman Optimality Equation

Bellman Optimality Equation for Q-values

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')]$$

We no longer need to search over all policies, only over all actions recursively!

Action selection: $\pi(s) \doteq \arg\max q_\pi(s,a)$

Calculating optimal Q-values: Q-Learning

[Watkins, 1989]

Q-learning

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Figure from: [Sutton & Barto, 2018]

Standard vs Inverse RL

UNI FREIBURG

Standard RL:

estimate optimal **policy**
from state, action, and reward sequences

$$(s_t, a_t, s_{t+1}, r_{t+1}, \ldots, s_{t+n}, r_{t+n})$$

↓ learn

$$\pi_\theta(s)$$

Environment

↓

Rewards → RL → Behavior

Inverse RL:

estimate unknown **reward** function
from state and action sequences

$$(s_t, a_t, s_{t+1}, a_{t+1}, \ldots, s_{t+n})$$

↓ learn

$$r_\psi(s, a)$$

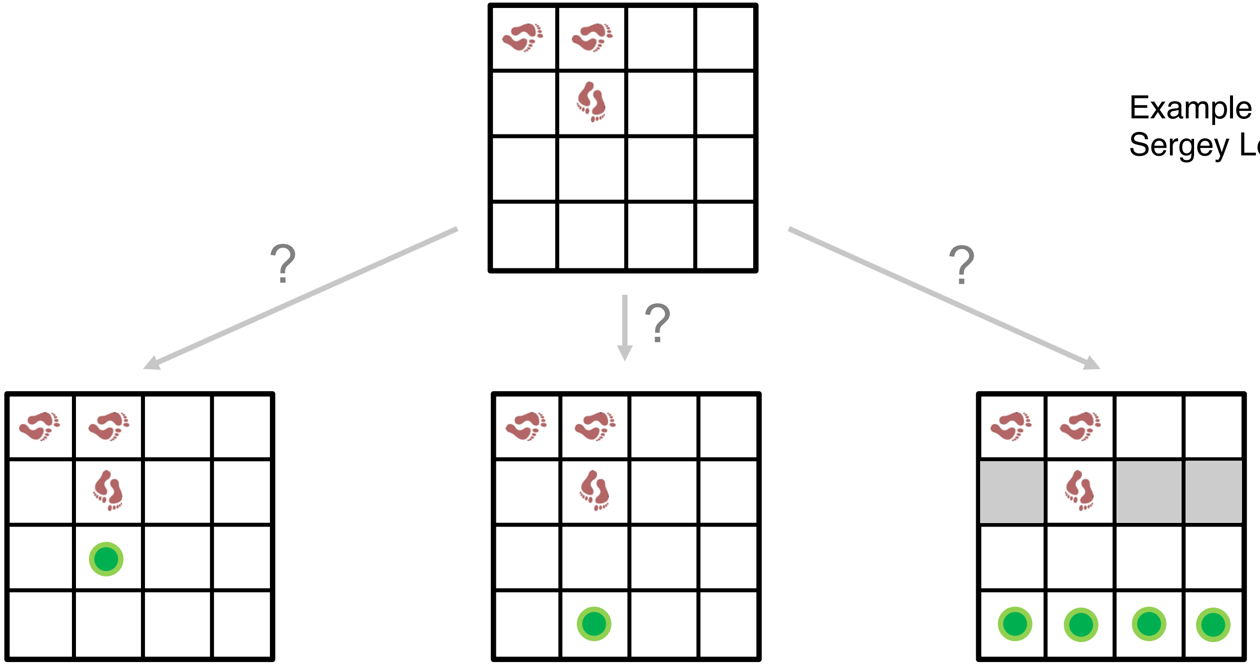
Environment

↓

Rewards ← IRL ← Behavior

Can we learn rewards from behavioral data?

Example credits:
Sergey Levine



The problem is underspecified: many reward functions would explain the behavior!

Can we learn rewards from behavioral data?

The problem is underspecified: many reward functions would explain the behavior!

Idea:
Account for uncertainty in the reward function by assuming a probabilistic behavior model that keeps action distribution in the policy as broad (non-committed) as possible

⇨ Maximum Entropy Inverse Reinforcement Learning

Problem: Needs to solve a full RL problem to convergence in the inner loop!

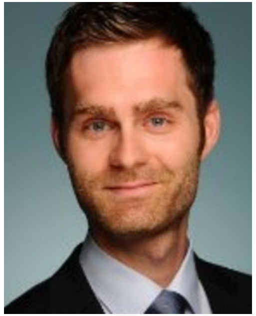
Deep Inverse Q-Learning (with Constraints)

[G. Kalweit, M. Huegle, M. Werling, J. Boedecker, NeurIPS, 2020]

Joint work with:

Gabriel Kalweit

Maria Kalweit



Moritz Werling

Deep Inverse Q-Learning

[G. Kalweit, M. Huegle, M. Werling, J. Boedecker, NeurIPS, 2020]

Probabilistic behavior assumption for the expert (here for two actions, *a* and *b*):

$$\frac{\exp(Q^*(s,a))}{\exp(Q^*(s,a)) + \exp(Q^*(s,b))} = \pi^{\mathcal{E}}(a|s) \quad \text{and} \quad \frac{\exp(Q^*(s,b))}{\exp(Q^*(s,a)) + \exp(Q^*(s,b))} = \pi^{\mathcal{E}}(b|s)$$

$$\Rightarrow \quad \exp(Q^*(s,a)) + \exp(Q^*(s,b)) = \frac{\exp(Q^*(s,a))}{\pi^{\mathcal{E}}(a|s)} = \frac{\exp(Q^*(s,b))}{\pi^{\mathcal{E}}(b|s)}$$

$$\Rightarrow \quad \exp(Q^*(s,a)) = \frac{\pi^{\mathcal{E}}(a|s)}{\pi^{\mathcal{E}}(b|s)} \exp(Q^*(s,b))$$

Taking logs: $Q^*(s,a) = Q^*(s,b) + \log(\pi^{\mathcal{E}}(a|s)) - \log(\pi^{\mathcal{E}}(b|s))$

Deep Inverse Q-Learning

[G. Kalweit, M. Huegle, M. Werling, J. Boedecker, NeurIPS, 2020]

UNI FREIBURG

$$Q^*(s,a) = Q^*(s,b) + \log(\pi^{\mathcal{E}}(a|s)) - \log(\pi^{\mathcal{E}}(b|s))$$

Using:

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} \mathbf{E}_{s' \sim \mathcal{M}(s,a,s')}[Q^*(s',a')]$$

and replacing the Q-values above to solve for the immediate reward leads to:

$$\begin{aligned} r(s,a) = \log(\pi^{\mathcal{E}}(a|s)) - \gamma \max_{a'} \mathbf{E}_{s' \sim \mathcal{M}(s,a,s')}[Q^*(s',a')] + r(s,b) \\ - \log(\pi^{\mathcal{E}}(b|s)) + \gamma \max_{b'} \mathbf{E}_{s' \sim \mathcal{M}(s,b,s')}[Q^*(s',b')]. \end{aligned}$$

Intuitively: immediate reward encodes the local probability of action *a* while also ensuring the probability of the maximizing next action *a'* under Q-learning

Deep Inverse Q-Learning

[G. Kalweit, M. Huegle, M. Werling, J. Boedecker, NeurIPS, 2020]

UNI FREIBURG

Defining: $\eta_s^a := \log(\pi^{\mathcal{E}}(a|s)) - \gamma \max_{a'} \mathbf{E}_{s' \sim \mathcal{M}(s,a,s')}[Q^*(s',a')]$

After some manipulation, the reward for *n* actions can be derived as:

$$r(s,a) = \eta_s^a + \frac{1}{n-1} \sum_{b \in \mathcal{A}_{\bar{a}}} r(s,b) - \eta_s^b.$$

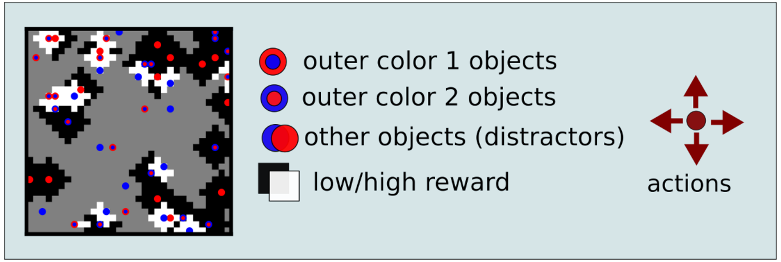
This leads to three novel algorithms:

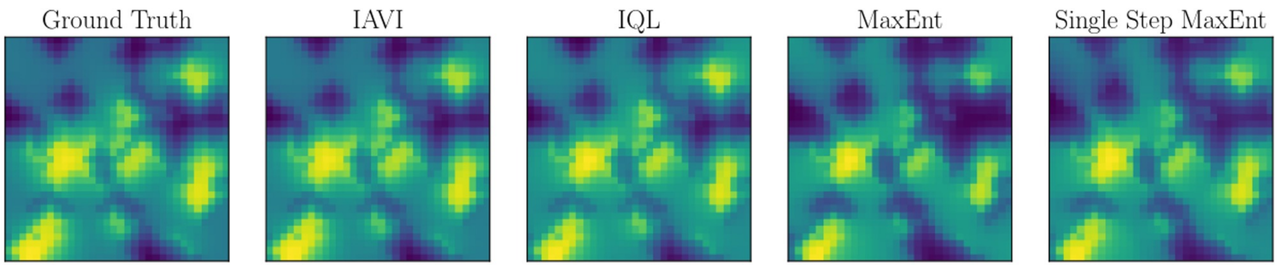
Inverse Action-Value Iteration **IAVI**	Tabular (Constrained) Inverse Q-Learning **(C)IQL**	Deep (Constrained) Inverse Q-Learning **D(C)IQL**
discrete state-spaces, model-based, non-linear rewards	discrete state-spaces, sampling-based, non-linear rewards	continuous state-spaces, sampling-based, non-linear rewards

Inverse Q-Learning: Results

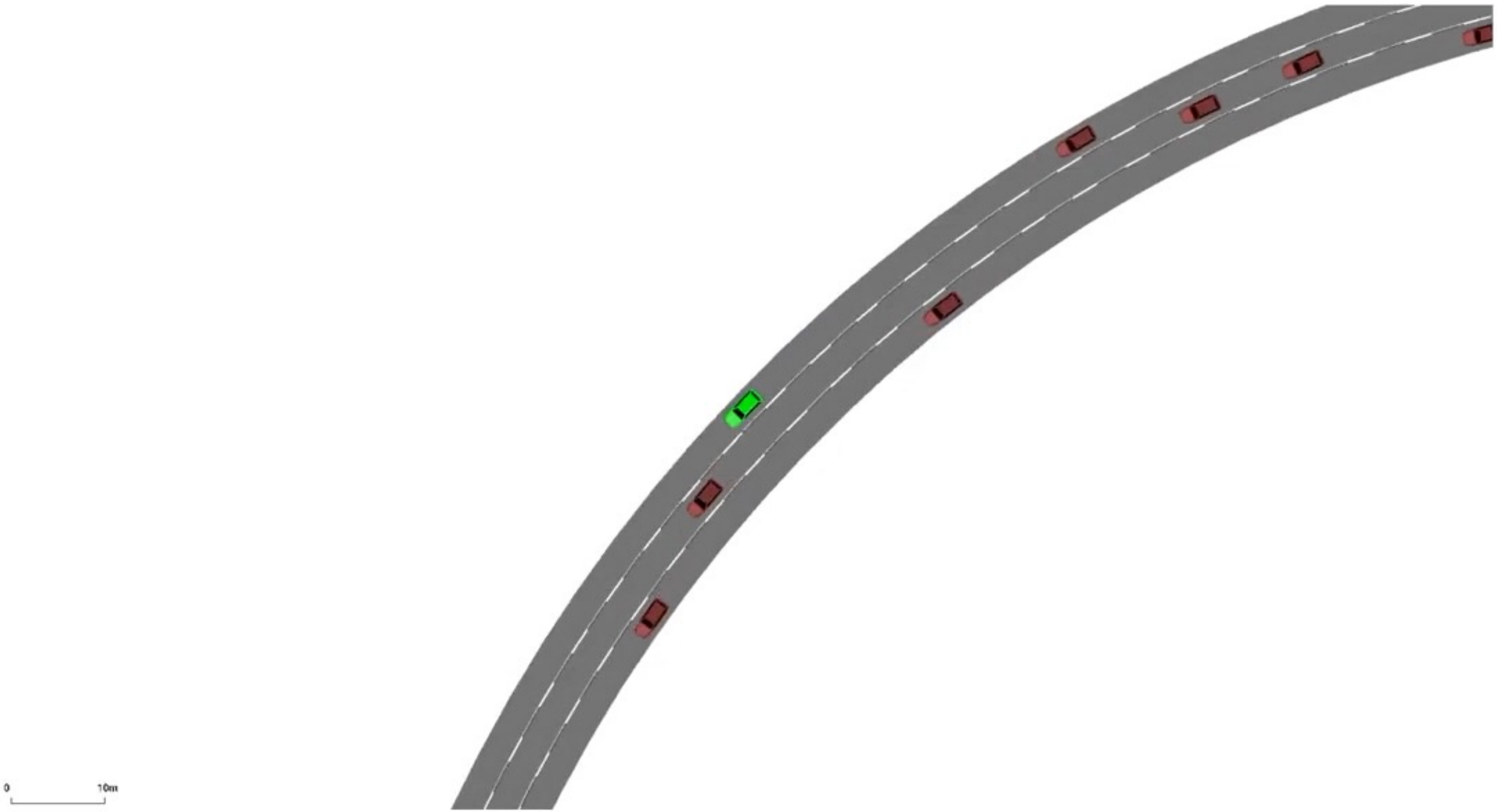
[G. Kalweit, M. Huegle, M. Werling, J. Boedecker, NeurIPS, 2020]

Toy-Benchmark:
Objectworld





	IAVI	IQL	MaxEnt	Single Step MaxEnt
EVD	**0.09 ± 0.00**	**1.47 ± 0.14**	11.58 ± 0.00	4.33 ± 0.00
Runtime	**0.03 ± 0.0 h**	**0.35 ± 0.0 h**	8.08 ± 1.0 h	12.2 ± 0.8 h



Expert Demonstrations
on US Highway

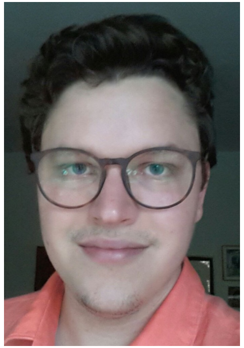
Closed-form Inverse RL for Neural Decoding

[Kalweit et al., ICML Comp Bio WS, 2021]

// BrainLinks BrainTools

UNI FREIBURG

Joint work with:



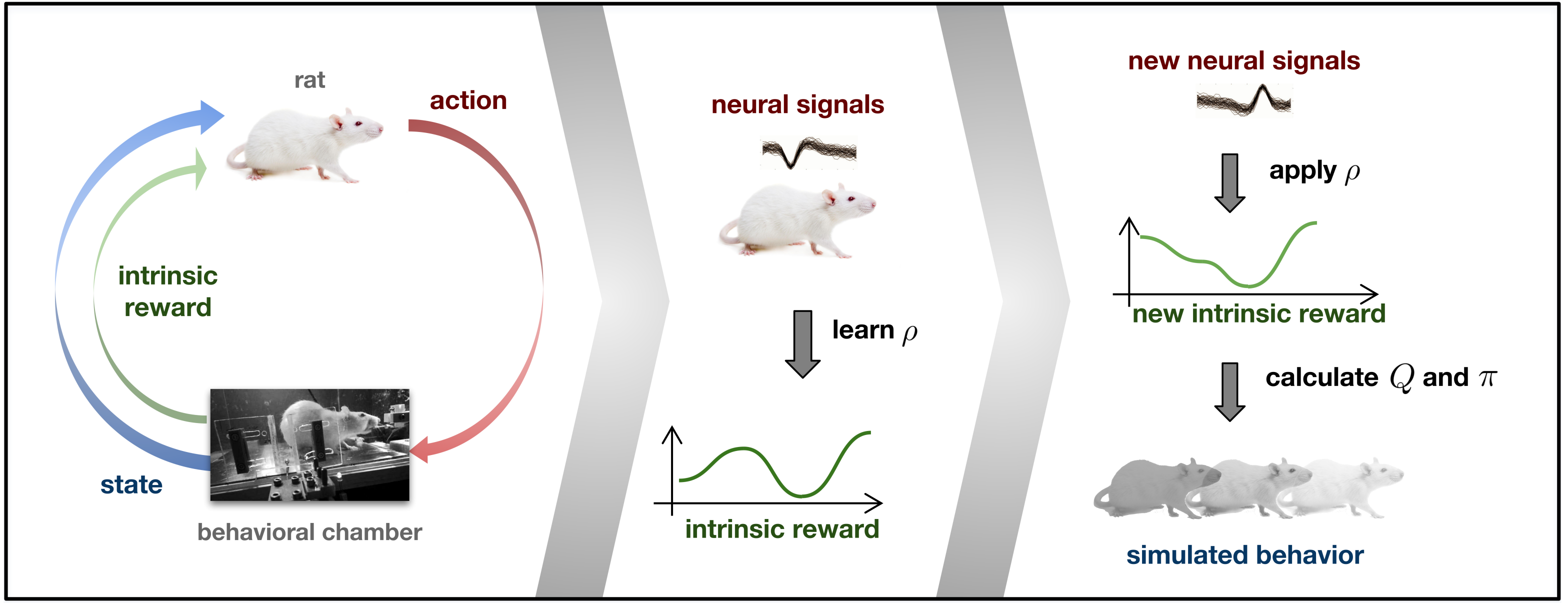
Gabriel Kalweit

Maria Kalweit (Hügle)

Ilka Diester

Mansour Alyahyay

Approach Summary

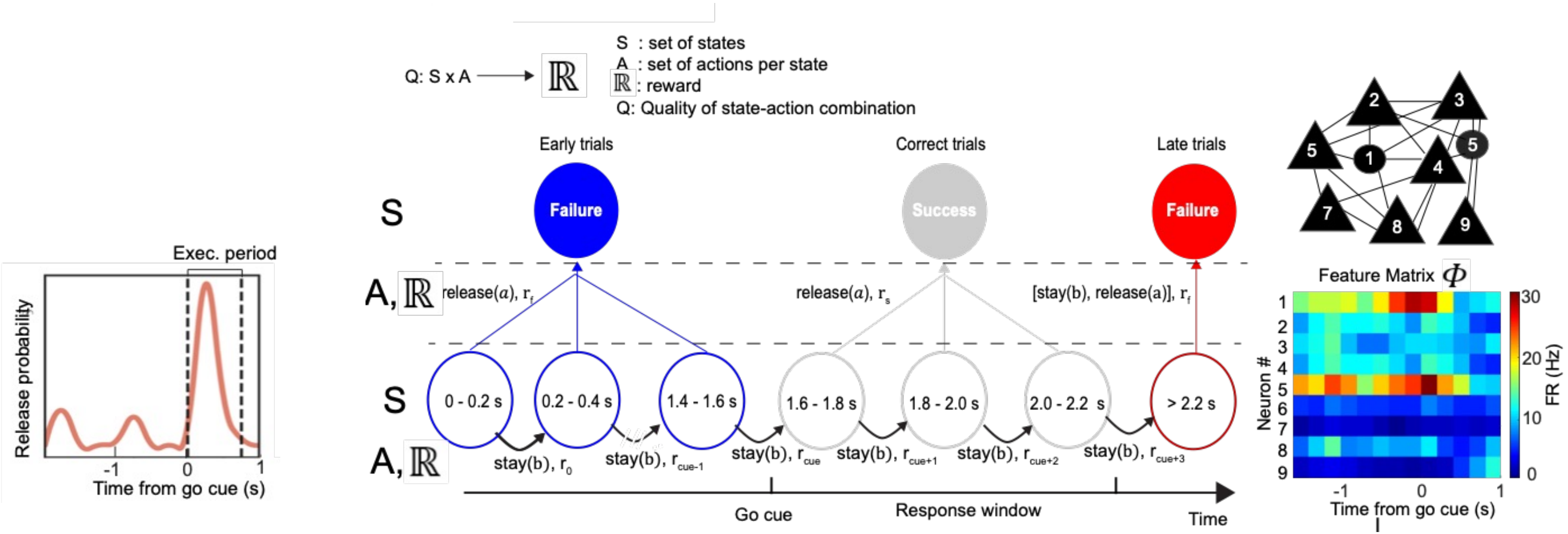


① Infer intrinsic reward from recorded trajectories via inverse Q-learning

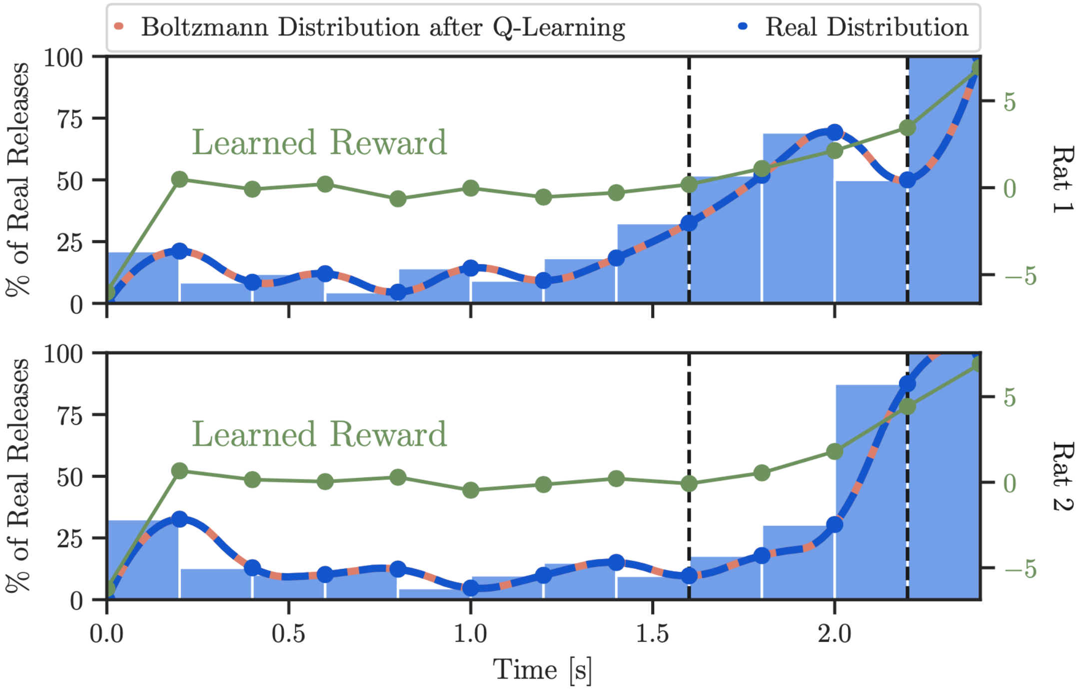
② Map neural signals to intrinsic reward

③ Predict behavior via intrinsic reward function and Q-learning

Training


Q: S x A → ℝ
S : set of states
A : set of actions per state
ℝ : reward
Q: Quality of state-action combination
Exec. period
Release probability
-1
0
1
Time from go cue (s)
Early trials
Correct trials
Late trials
S
Failure
Success
Failure
A, ℝ
release(a), r_f
release(a), r_s
[stay(b), release(a)], r_f
S
0 - 0.2 s
0.2 - 0.4 s
1.4 - 1.6 s
1.6 - 1.8 s
1.8 - 2.0 s
2.0 - 2.2 s
> 2.2 s
A, ℝ
stay(b), r_0
stay(b), r_cue-1
stay(b), r_cue
stay(b), r_cue+1
stay(b), r_cue+2
stay(b), r_cue+3
Go cue
Response window
Time
Feature Matrix Φ
Neuron #
FR (Hz)
Time from go cue (s)

Reward Estmation via IAVI



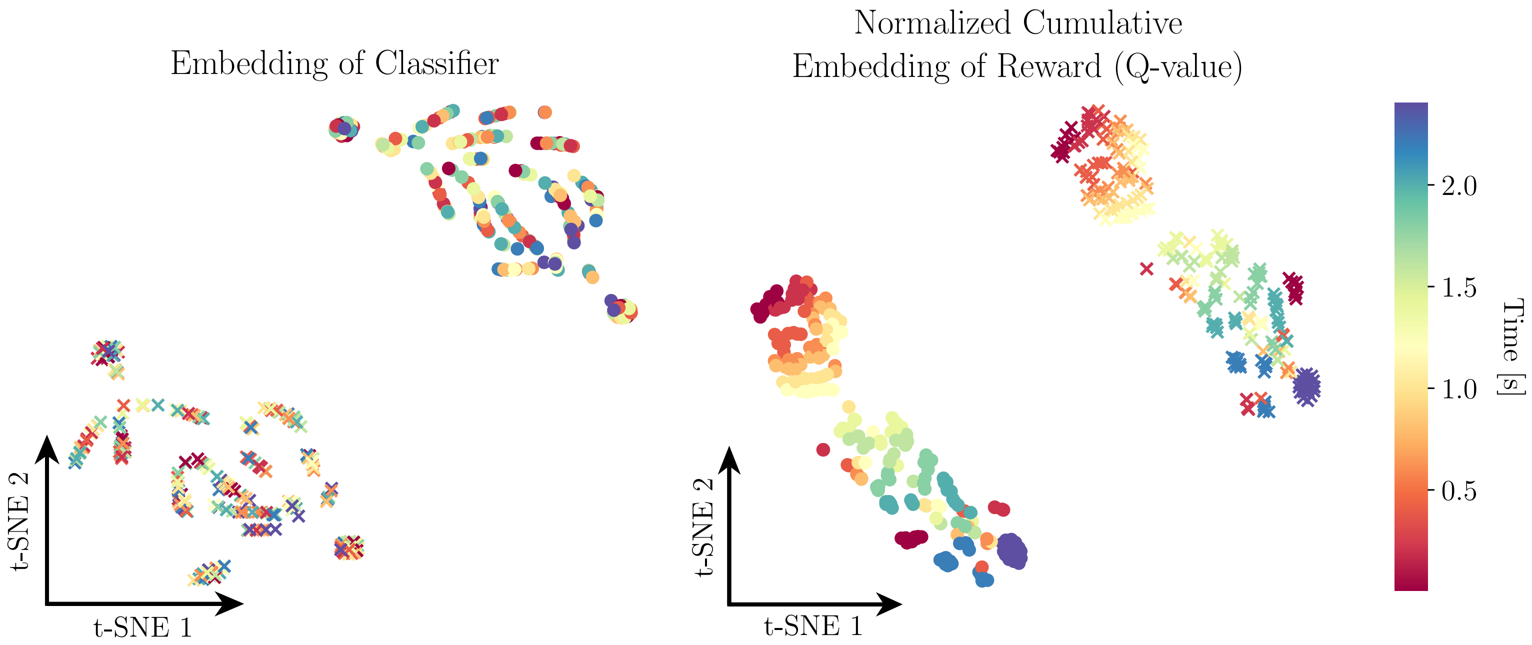
IAVI returns a **scalar reward function** precisely **encoding the recorded behavior** as an intermediate result, which can then be used for neural decoding

Per-Trial Release Behavior Prediction

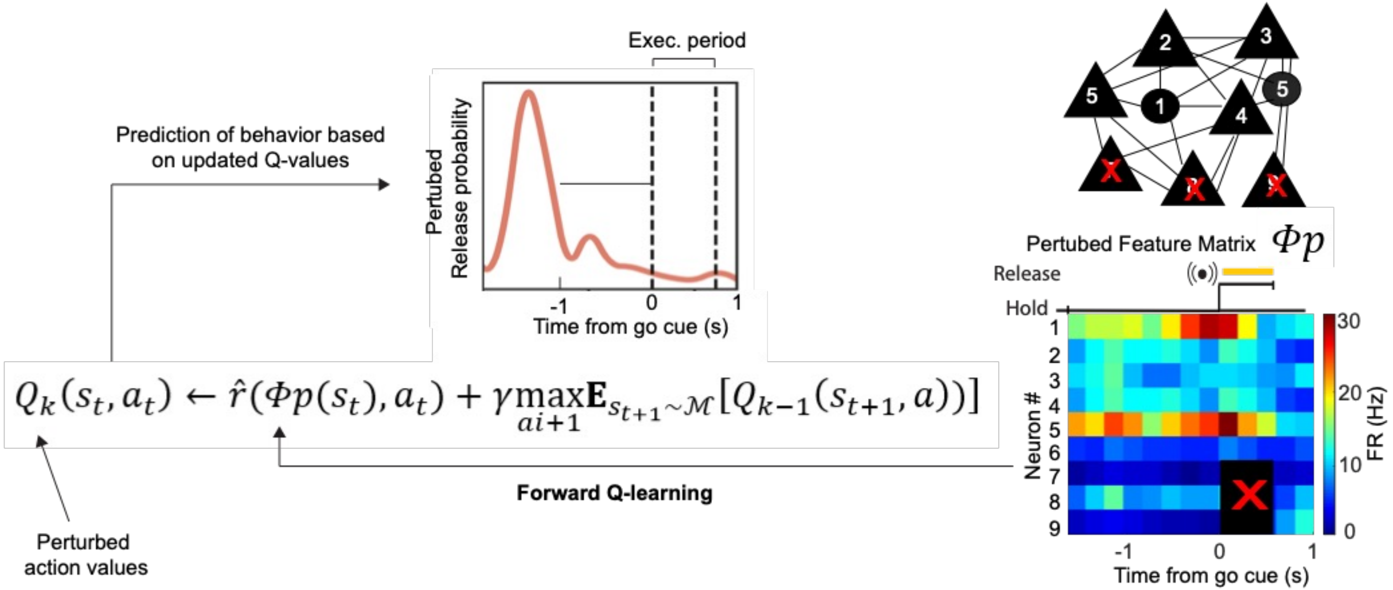
- Compare **NeuRL (ours)** to a random controller, logistic regression **(LR)** and non-linear classification via neural-networks **(NNC)**
- For NeuRL and NNC, **optimize hyperparameters** with random search with 500 sampled configurations each
- Consider **release prediction** in a trial if controller assigns a **probability of > ε** (here ε = 0.6) to action release in a given time step
- Results evaluated using **10-fold cross-validation** on a **test set**

	Rat 1			Rat 2		
	Exact Match	Near 1 Match	Near 2 Match	Exact Match	Near 1 Match	Near 2 Match
NeuRL	**0.36**(±**0.11**)	**0.49**(±**0.13**)	**0.59**(±**0.09**)	**0.44**(±**0.09**)	**0.62**(±**0.06**)	**0.70**(±**0.11**)
NNC	0.21(±0.09)	0.28(±0.12)	0.37(±0.17)	0.34(±0.10)	0.46(±0.09)	0.52(±0.10)
LR	0.15(±0.07)	0.19(±0.10)	0.29(±0.08)	0.33(±0.09)	0.41(±0.08)	0.47(±0.10)
Random	0.04(±0.07)	0.2(±0.13)	0.29(±0.15)	0.12(±0.06)	0.38(±0.07)	0.46(±0.10)

Visualization of Latent Embeddings

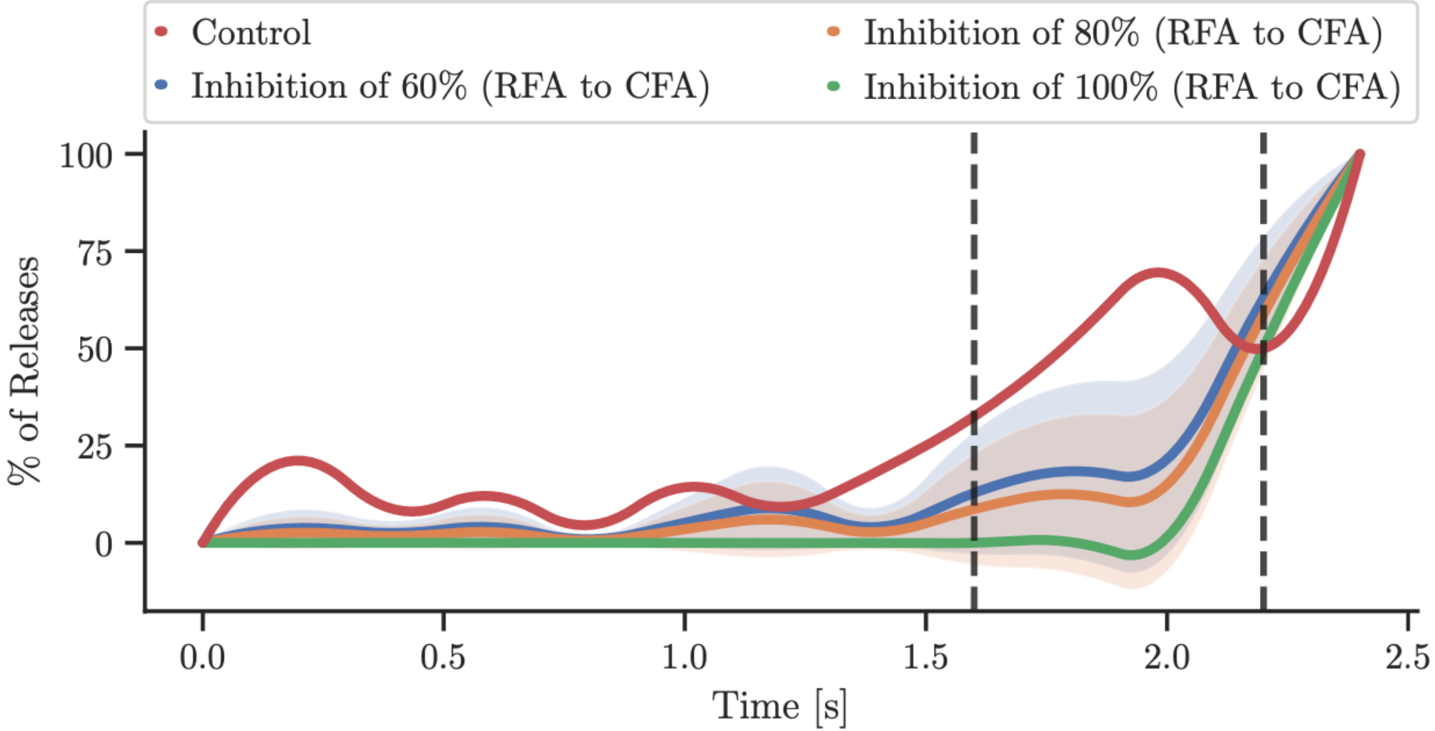


Testing

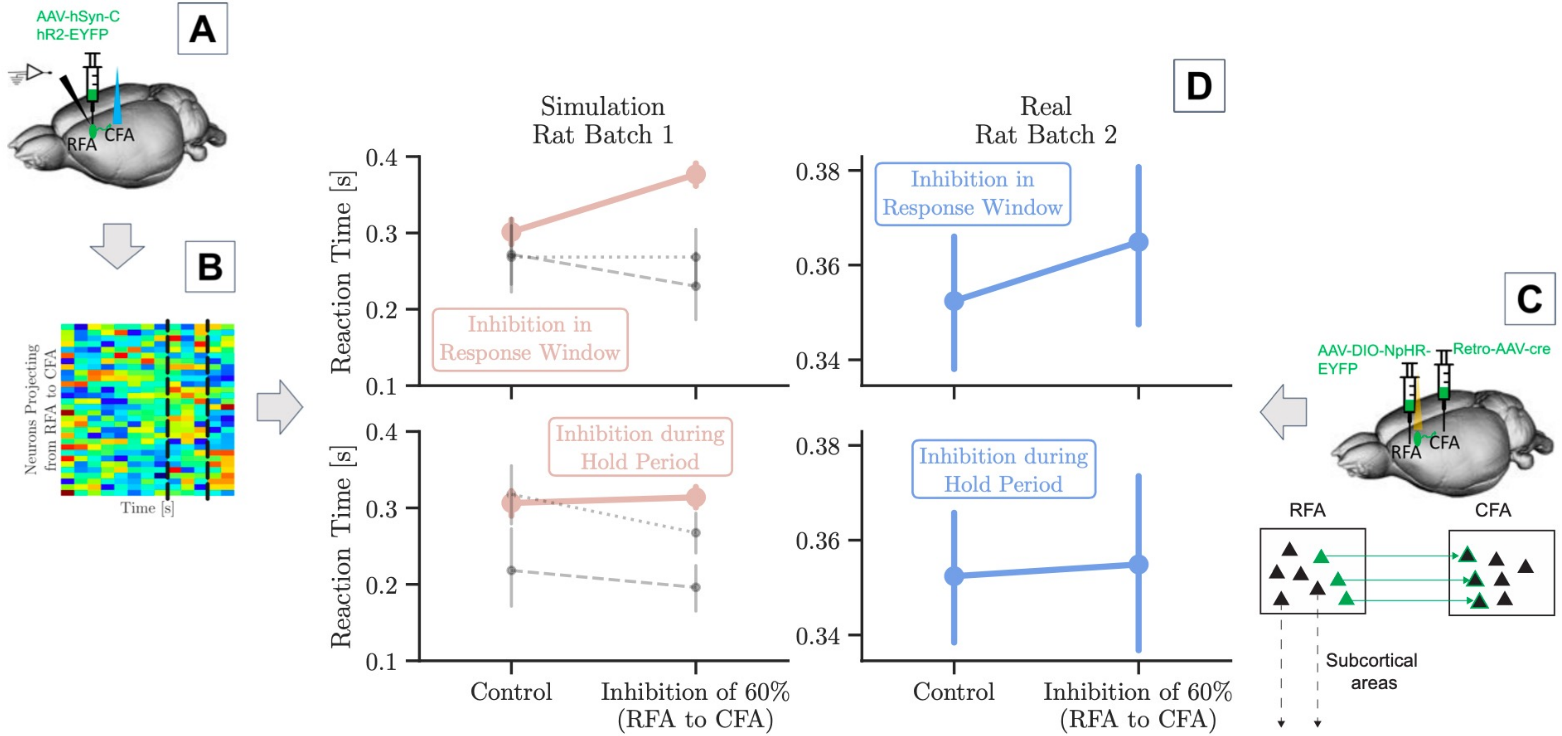


Prediction of behavior based on updated Q-values
Exec. period
Pertubed
Release probability
-1
0
1
Time from go cue (s)
2
3
5
1
4
5
Pertubed Feature Matrix Φp
Release
Hold
Neuron #
1
2
3
4
5
6
7
8
9
-1
0
1
Time from go cue (s)
30
20
10
0
FR (Hz)
$Q_k(s_t,a_t) \leftarrow \hat{r}(\Phi p(s_t),a_t) + \gamma \max_{ai+1} \mathbf{E}_{s_{t+1}\sim\mathcal{M}}[Q_{k-1}(s_{t+1},a))]$
Forward Q-learning
Perturbed
action values

Inhibition Simulation I

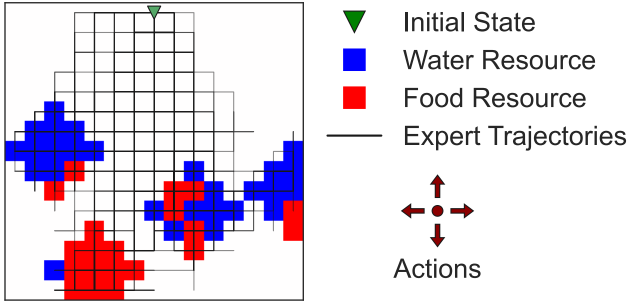


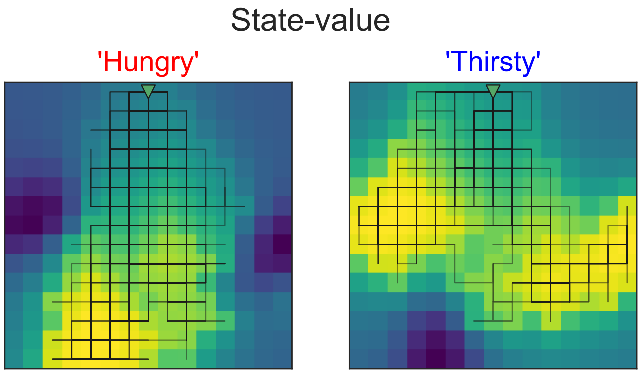
Inhibition Simulation II

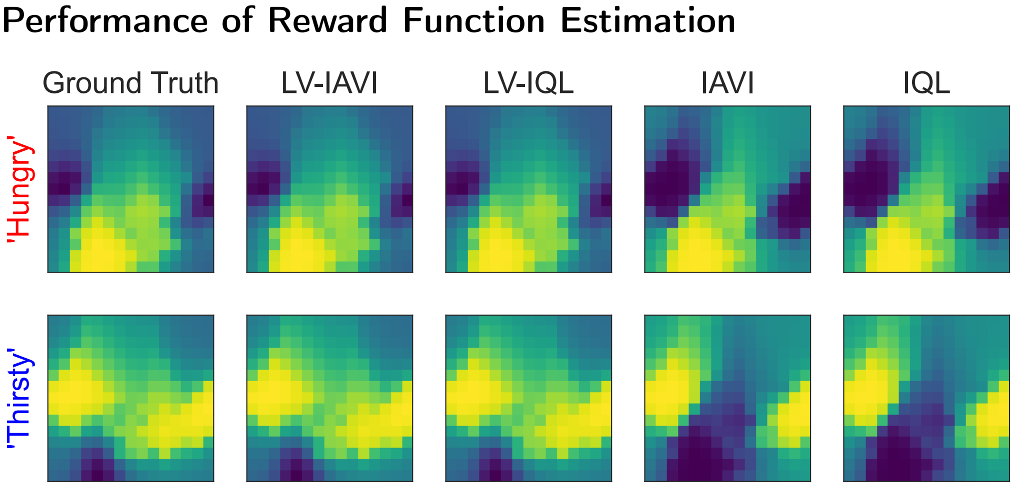


Outlook: IQL with Multiple Intentions

Recent (exciting!) extension by Hao Zhu







		LV-IAVI	LV-IQL	IAVI	IQL
EVD	'Hungry'	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	21.42 ± 0.00	21.38 ± 0.00
	'Thirsty'	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	38.00 ± 0.00	38.05 ± 0.00

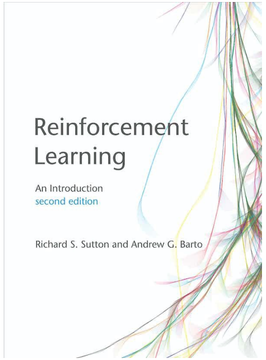
Exercises

Colab Notebook:

https://colab.research.google.com/drive/1YbHB0V1JQ5e_0T5zIR-nmRwmNOILY6v-?usp=sharing

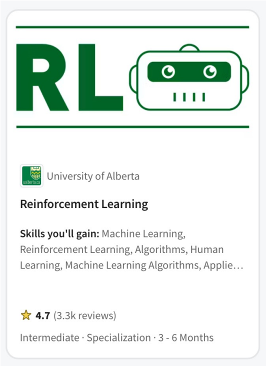
Complete and play around with Value Iteration and Q-Learning for different tasks

Further Resources



Standard RL text book (very accessible, free PDF):

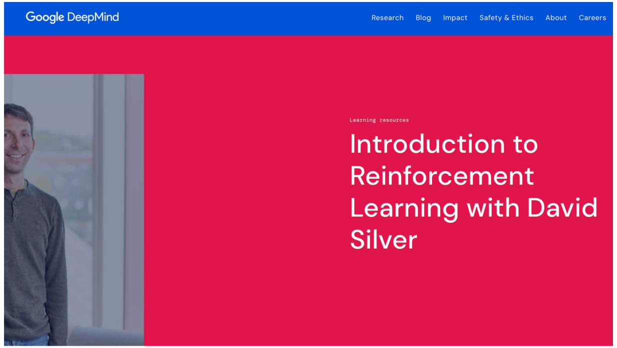
http://incompleteideas.net/book/RLbook2020.pdf



Nice 4-part course on coursera:

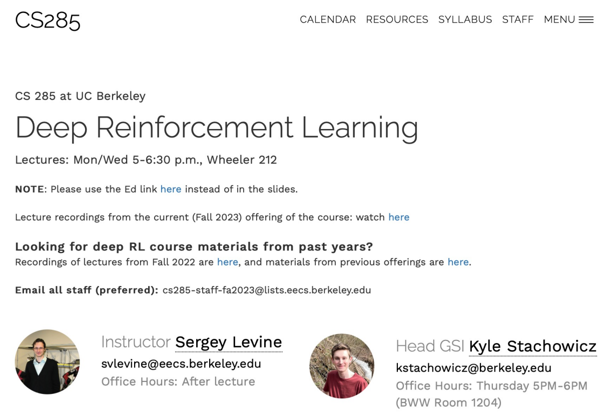
https://www.coursera.org/specializations/reinforcement-learning

Further Resources



Very good “classic” RL course:

https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver



Very comprehensive (Deep) RL course at UC Berkeley:

http://rail.eecs.berkeley.edu/deeprlcourse/