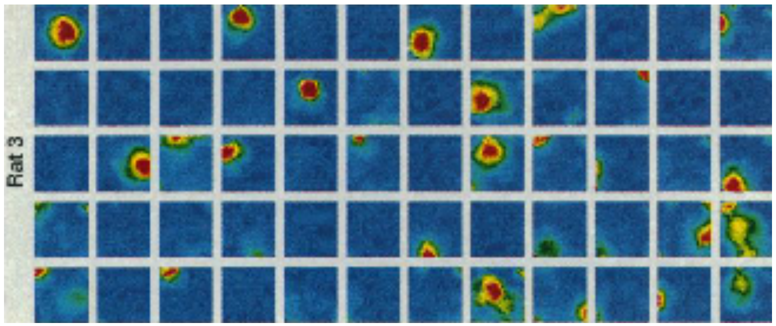
Prioritized Sweeping Neural Dyna-Q with Multiple Predecessors

(and Hippocampal Replays)

Lise Aubin, Mehdi Khamassi, Benoît Girard

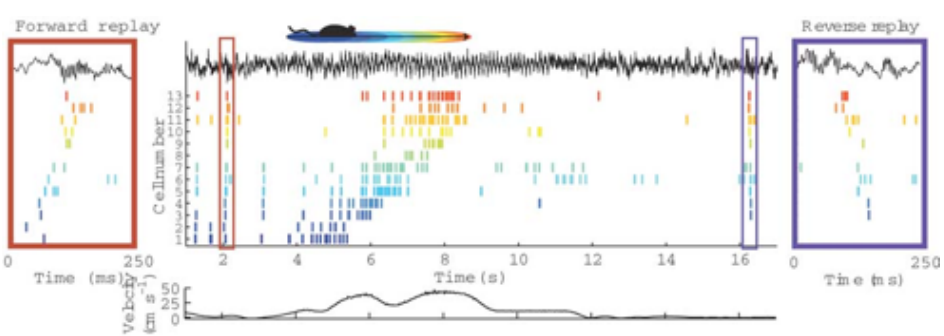
Context

- Place cells [O'Keefe, 1971] :



- Place field = encode the current position of the animal
- Cognitive map of the environment

- Reactivations in rats [Wilson & McNaughton, 1994]:

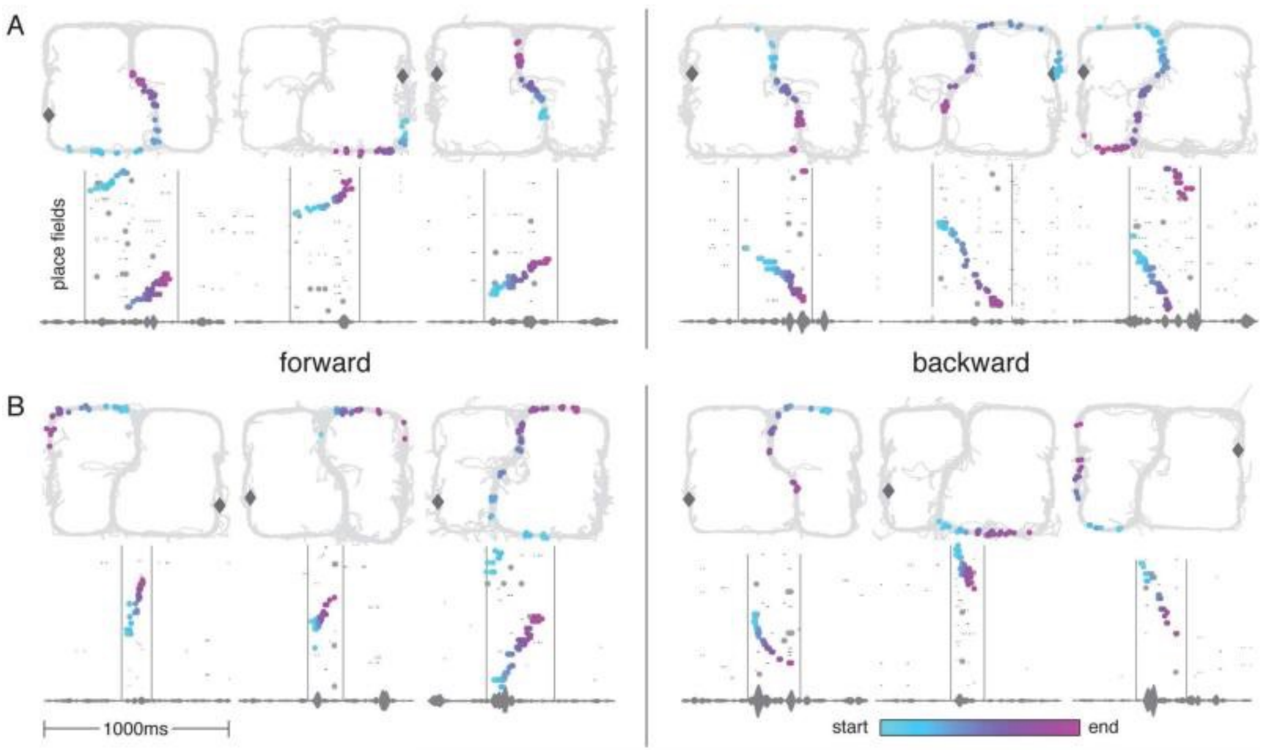


- Occurred during sleep or awake rest
- Majority of disordered replays
- Can be forward/backward/novel sequences
- Could correspond to memory consolidation (Buzsaki, 1989) / reinforcement learning (de Lavilléon et al., 2015)

Context

- Imaginary replays [Gupta et al., 2010] :

  - Reactivations at rest
  - backward, forward & imaginary



Objectives

Investigate whether RL algorithms can explain the experimentally observed replays

- TD learning algorithm used to explain navigation learning (Arléo & Gerstner 00, Foster et al. 00)
- TD learning convergence can be improved with dyna approaches (Sutton, 90):
  - World transition model
  - Offline simulation

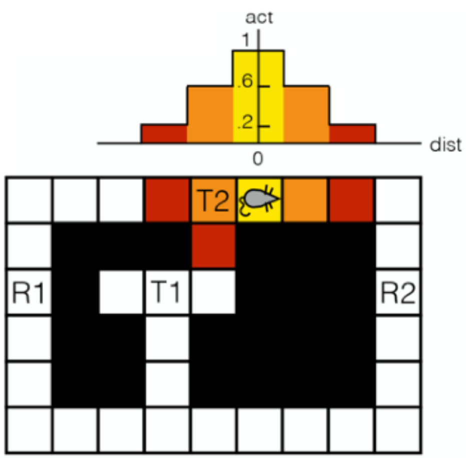
→ What if we have a limited offline budget?

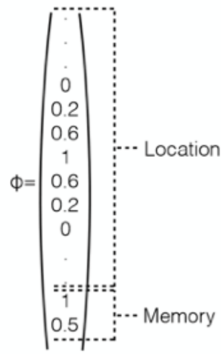
Dyna-Q with prioritized sweeping (Peng & Williams, 93; Moore & Atkeson, 93)

- The off-line replay phase has a fixed budget B
- The replayed states are chosen based on decreasing priority
- The priority of a state corresponds to either the reward prediction error (RPE) experienced on-line, or the predicted RPE (using an internal model of the transitions).

Simulated task

- Discrete version of the (Gupta et al., 2010) maze:





Training schedule:

1. Forced Turn Right (40 trials)
2. Forced Turn Left (40 trials)
3. Successions of Turn Right, Turn Left and alternate sessions

States description:

Place cell like activity
\+
Memory of the location of the last two reward events

Memory:

L	R
1	0
0	1
1	0.5
0.5	1

- 32 reachable squares
- 1 square = 1 place cell

- 1 place field: about 10cm
- 4 actions: North/South/East/West

Neural Dyna-Q with prioritized sweeping

3 two layers networks:

- N_Q^a = Q-value network
- N_R^a = Reward network
- N_P^a = Predecessors network

World model

Configuration:

Networks parameters		
Learning rate	0.1	0.1
Hidden Neurons	16	26
Weights initialization	0.1	0.0045
Sigmoid slope in hidden/output layer	1/0.4	1/0.5

Softmax policy:

$$P(a) = \frac{\exp \frac{Q_t(a)}{\tau}}{\sum_{b=1}^{n} \exp \frac{Q_t(b)}{\tau}}$$

Training:

Network	Input	Output
N_Q^a	ϕ^t	$r + \gamma \max_a (N_Q^a(\phi^{t+1}))$
N_R^a	ϕ^t	r
N_P^a	ϕ^{t+1}	ϕ^t

Priority: $|\delta| = |N_Q^a(\phi^t) - (N_R(\phi^t) + \gamma \max_a (N_Q^a(\phi^{t+1})))|$ Add $(\phi^t, |\delta|)$ in *PQueue*

B replay budget:

→ Pop the first element ϕ in *PQueue*,
→ for each action a :
    → Get its predecessor: $p = N_P^a(\phi)$
    → Update *Q-values*: train network N_Q^a:
        input = p -> output = $N_R(p) + \gamma \max_a (N_Q^a(\phi))$
    → Add predecessor in *PQueue* with priority:
        $|N_Q^a(p) - (N_R(p) + \gamma \max_a (N_Q^a(\phi)))|$

B

Our contribution

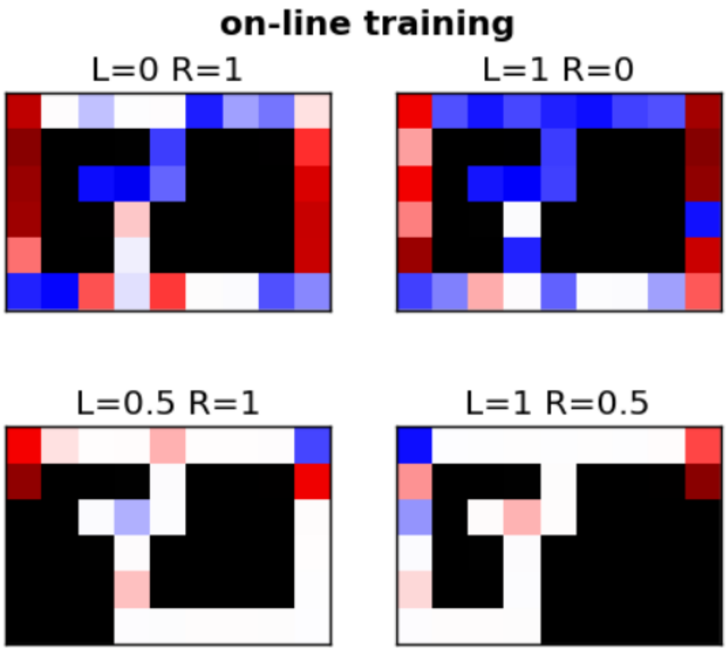
1. World model has to be learned off-line with shuffled training samples

2. An algorithm to train a network to associate one input to multiple outputs

3. Preliminary comparison between simulated and real replays

1. Online vs Off-line learning

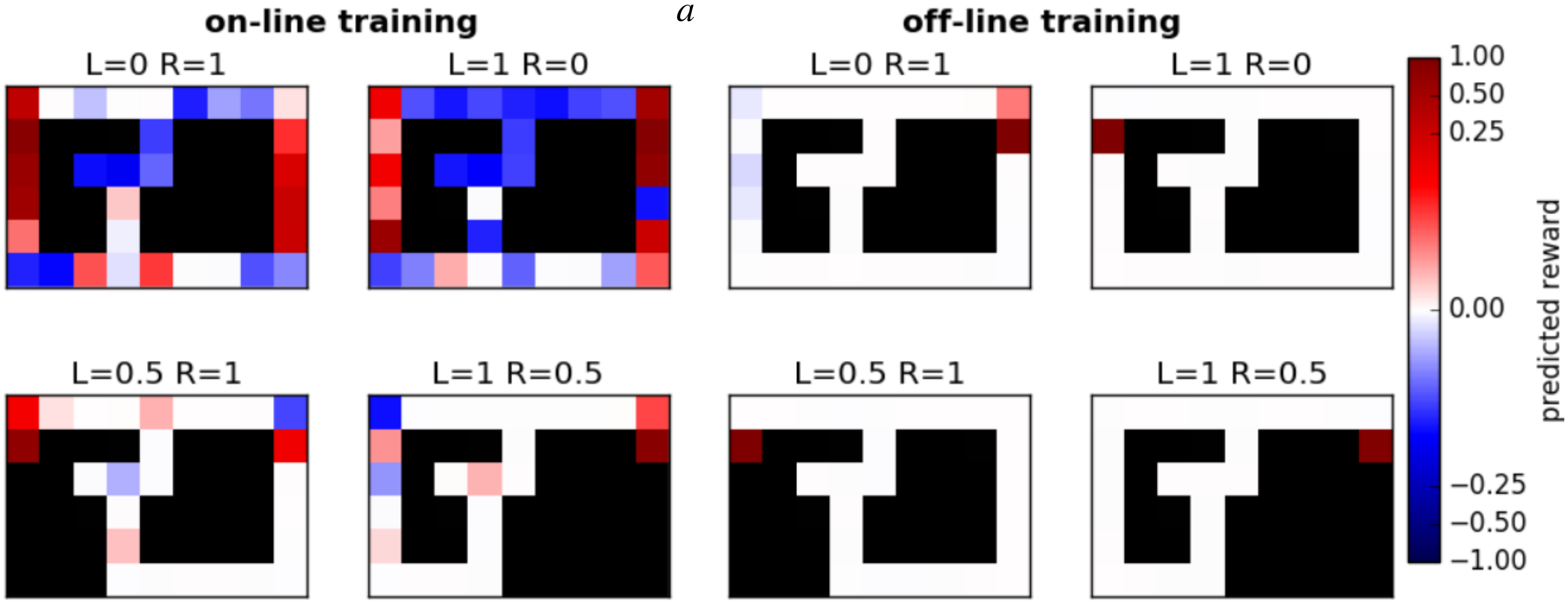
$$\max_a N_R^a$$



→ Strong correlation in the successive training samples prevents good on-line learning of the reward and transition models (reminiscent of Mnih et al. 2015)

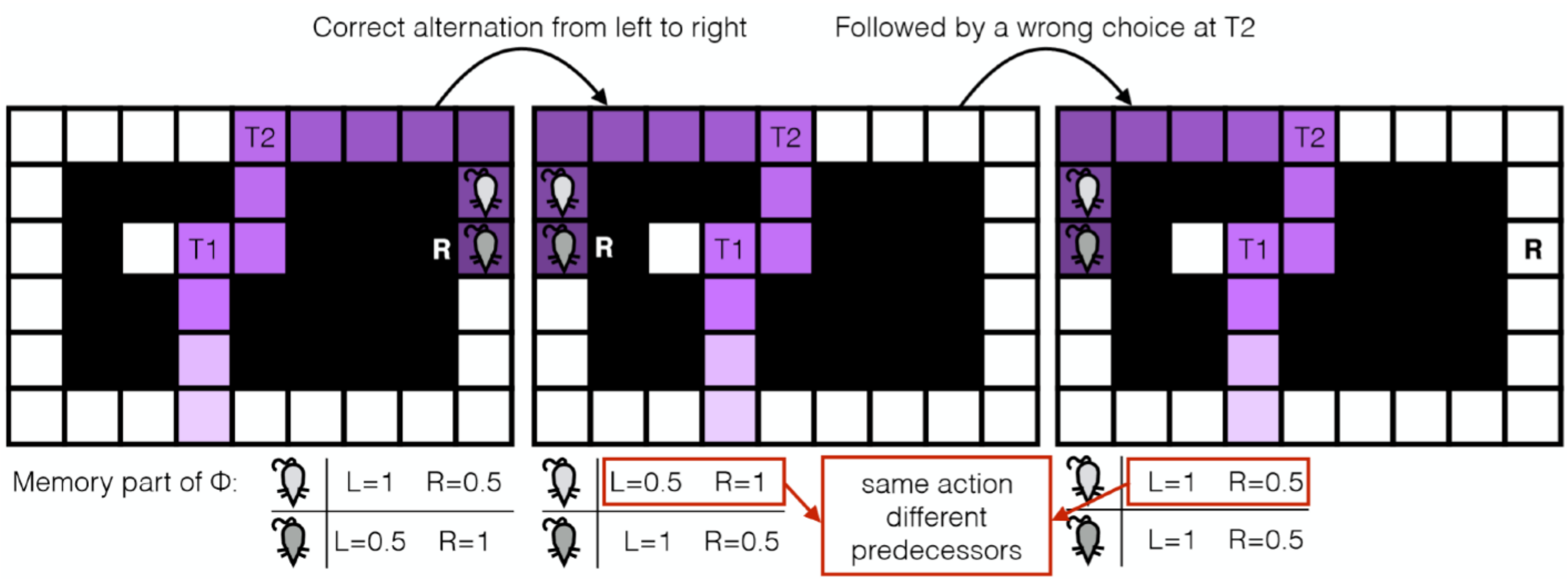
1. Online vs Off-line learning

$$\max_a N_R^a$$



→ Strong correlation in the successive training samples prevents good on-line learning of the reward and transition models (reminiscent of Mnih et al. 2015)

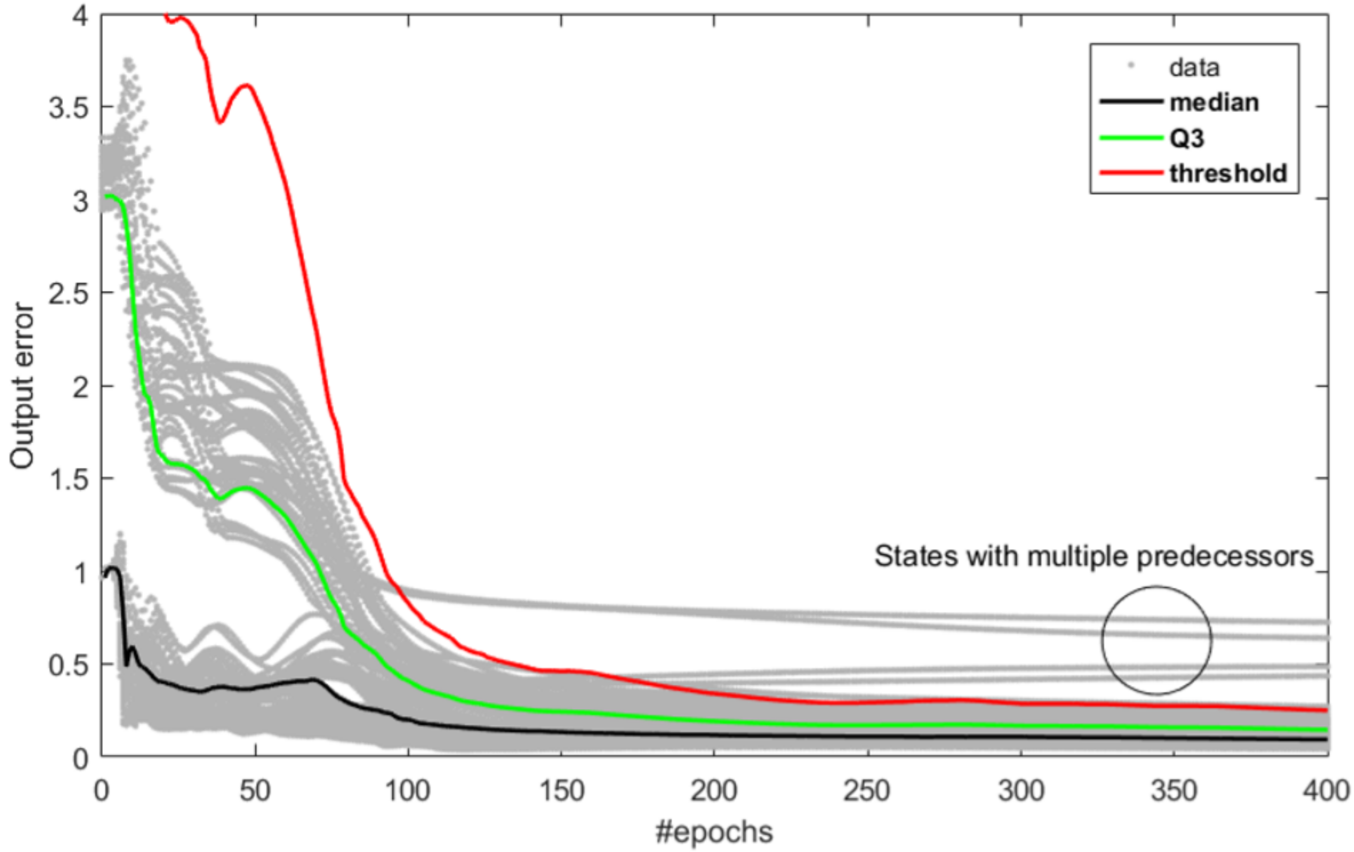
2. Multiple predecessor problem



→ one state can have **multiple predecessors**

2. Multiple predecessor problem

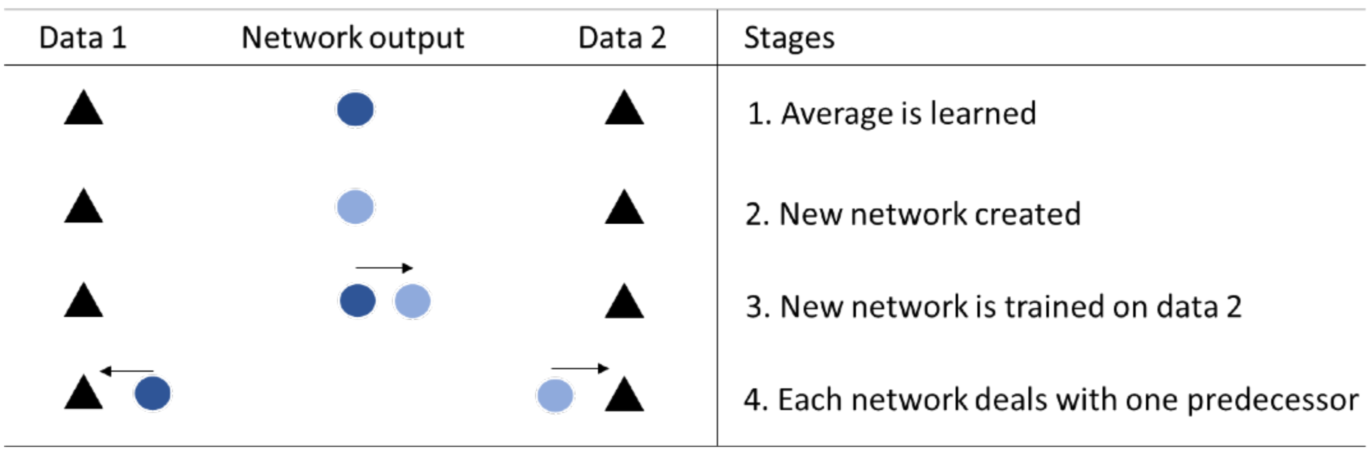
Predecessor network output error during learning



threshold = median + (Q3-median)*3

2. GALMO (Growing algorithm to learn multiple outputs)

Principle:

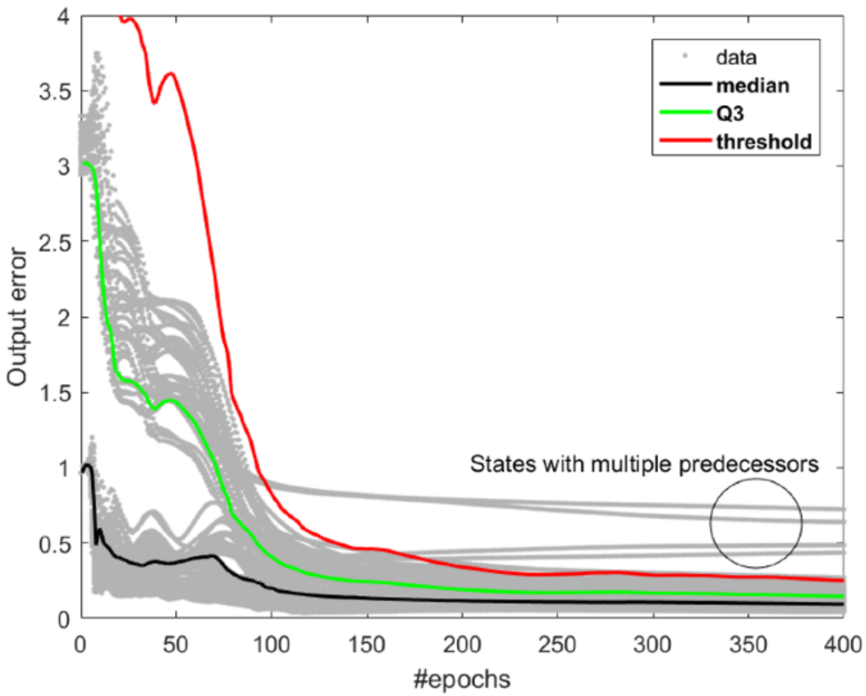


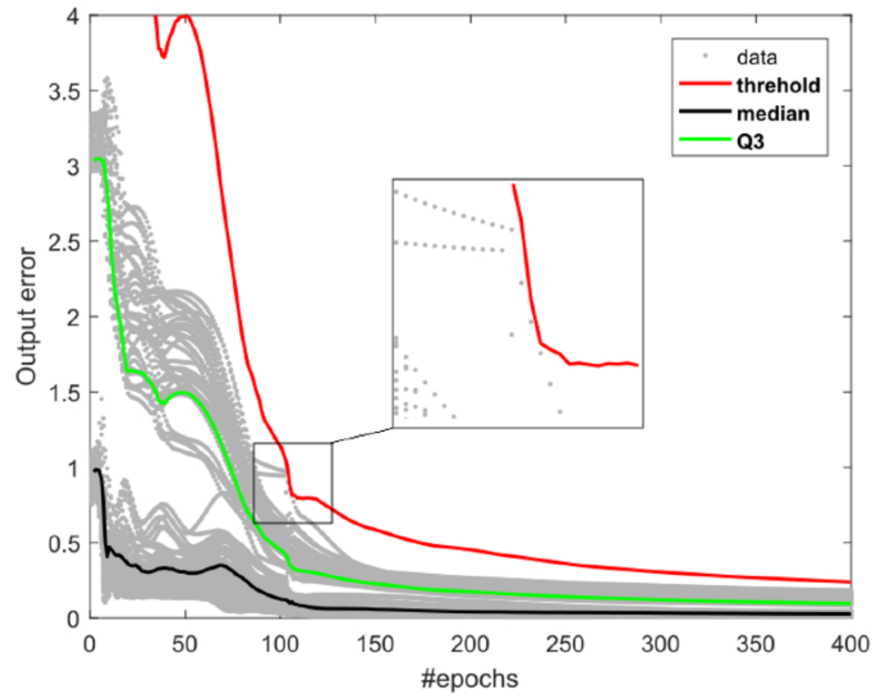
Replays:

- 1 list of predecessor networks $[N_P^a]$
- 1 list of « Gating Network » $[G_P^a]$ (one per N_P^a)

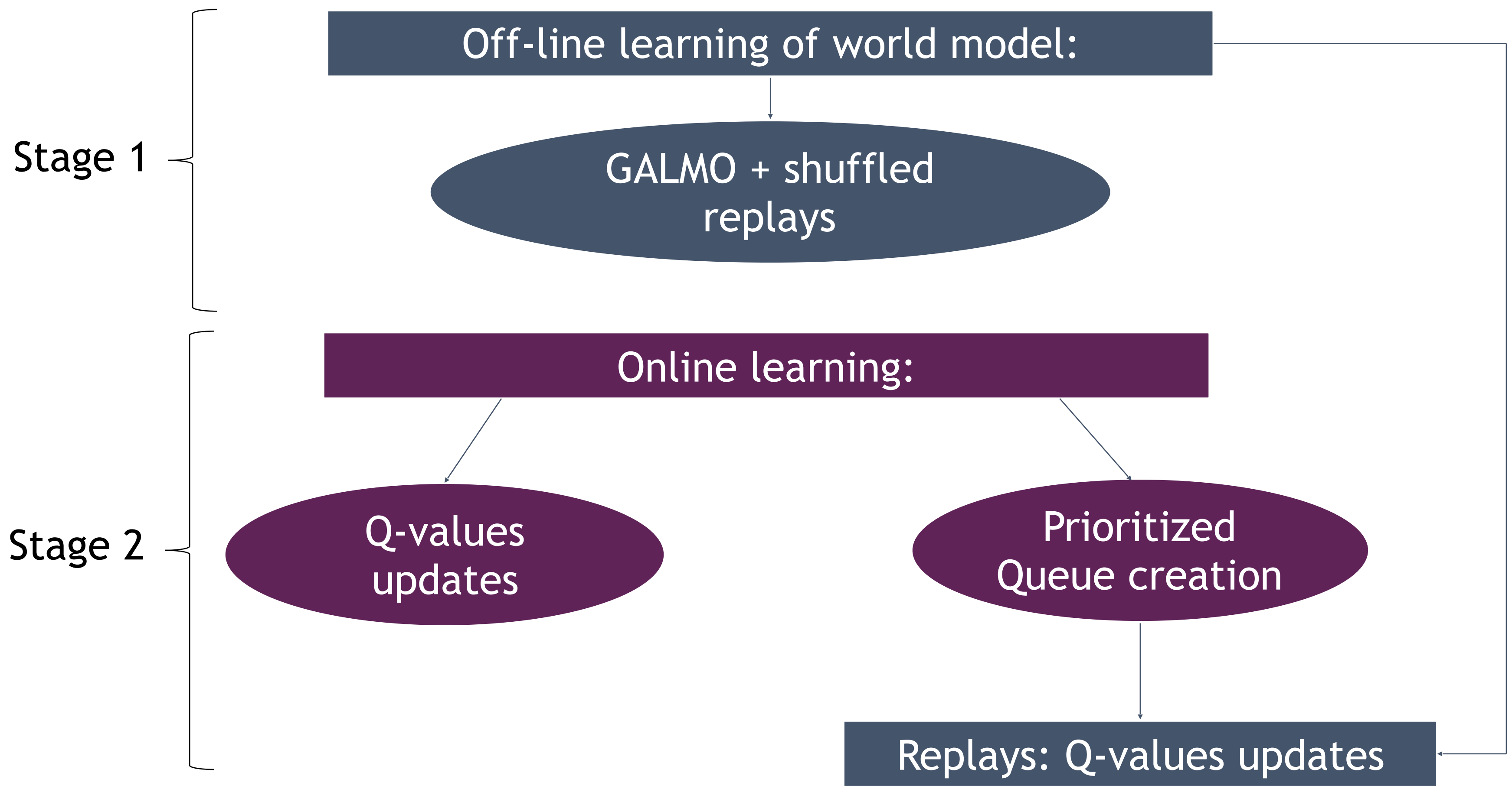
Gating Network = $\left\{ \begin{array}{l} \text{- Return 1 if } N_P^a \text{ network can give one predecessor} \\ \text{- Return 0 otherwise} \end{array} \right.$

2. Learning results with GALMO



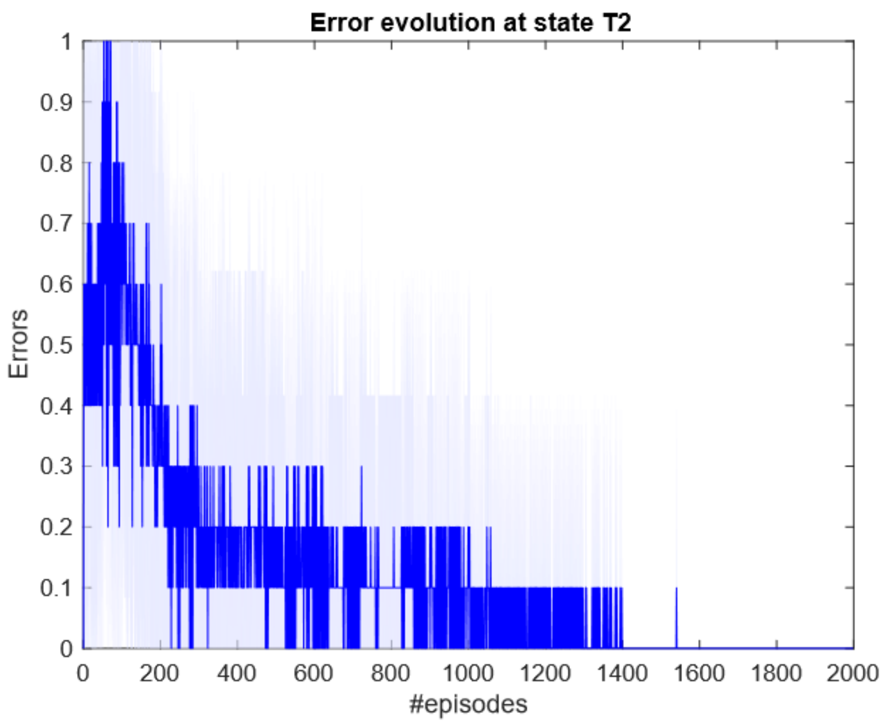


1+2. General Architecture

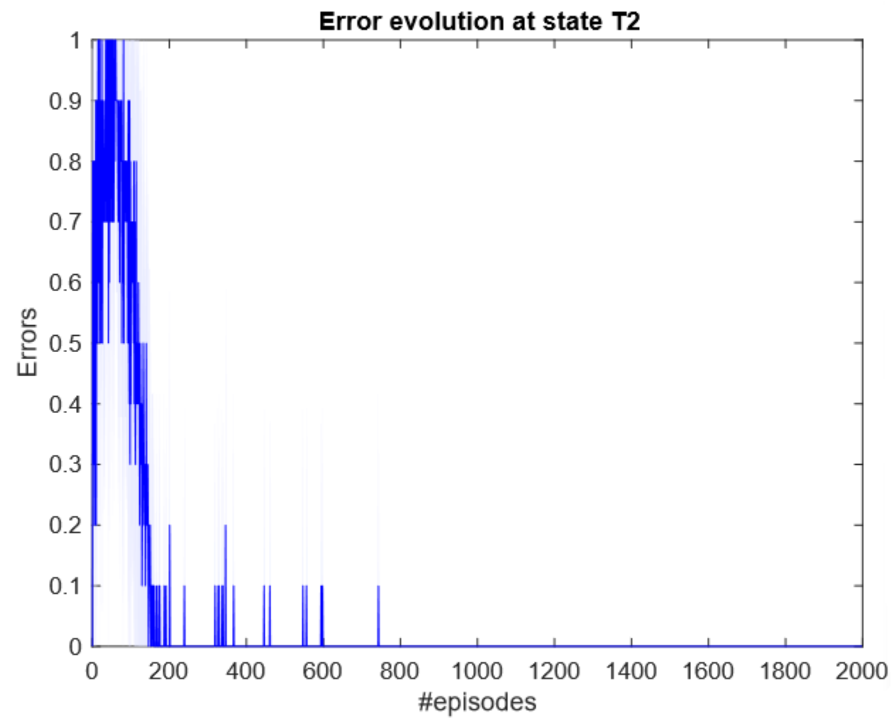


1+2. Without vs with replays

Q-learning:



Dyna-Q with prioritized sweeping:



Average for 10 runs

Average standard deviation

3. Replays generated by neural Dyna-Q with prioritized sweeping

What (Gupta et al., 2010) reported:	VS	What we generated:
• Similar amounts of backward and forward replays	≠	• Majority of unordered replays with: - significant amounts of backward replays - only occasional forward replays
• More opposite side replays in Turn Left and Turn Right than in Alternate	=	• More opposite side replays in Turn Left and Turn Right than in Alternate
• Novel shortcut sequences	≠	• No “novel sequences”

Conclusion

- Neural implementation of Dyna-Q with prioritized sweeping:

  → Two type of replays needed: | Learning world model -> **shuffled** replays
  Learning Q-values -> **prioritized** replays

- The **GALMO** algorithm is operational.

- The proposed prioritized sweeping implementation

… could explain:

✓ More opposite side replays in Turn Left and Turn Right than in Alternate

…could not explain:

❌ Similar amounts of backward and forward replays

❌ Novel sequences

→ Alternative RL algorithms have to be tested (alone or in conjunction with Dyna-Q).

→ Rats use multiple learning systems to learn navigation tasks (Khamassi and Humphries, 2012)

→ These systems generate different type of replays (Caze et al, in press)

The ReScience Journal

Reproducible Science is good.
Replicated Science is better.

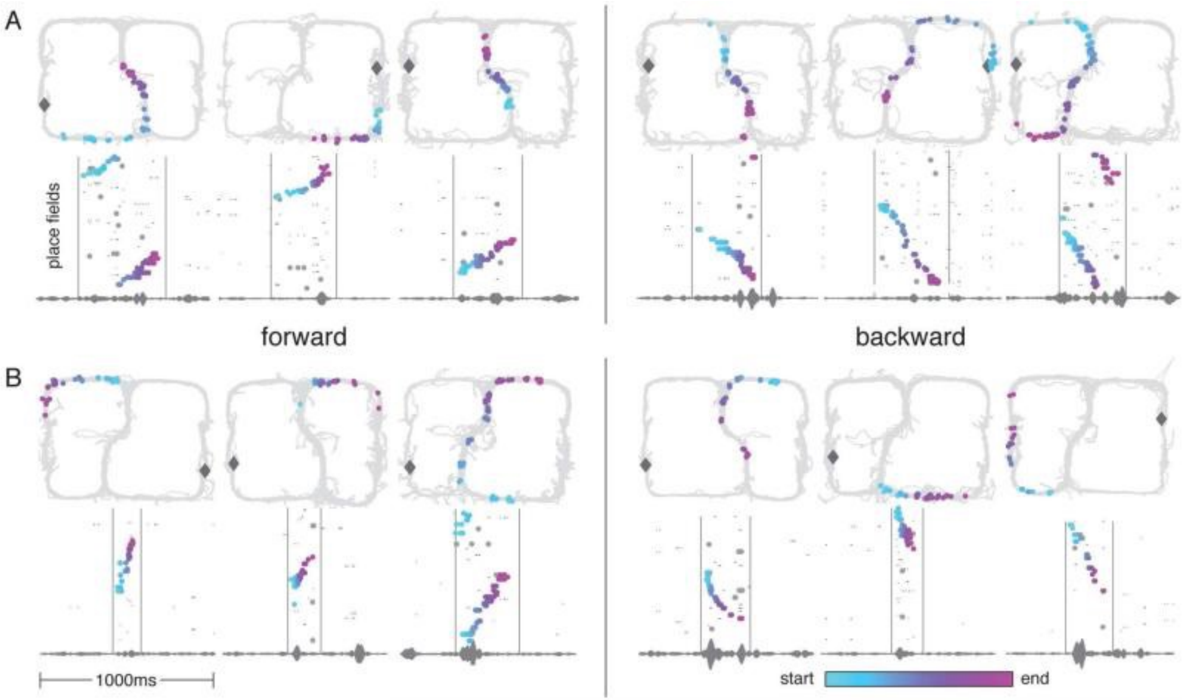
ReScience is a peer-reviewed journal
that targets computational research
and encourages the explicit replication of already published research, promoting new and open-source implementations in order to ensure that the original research is reproducible.

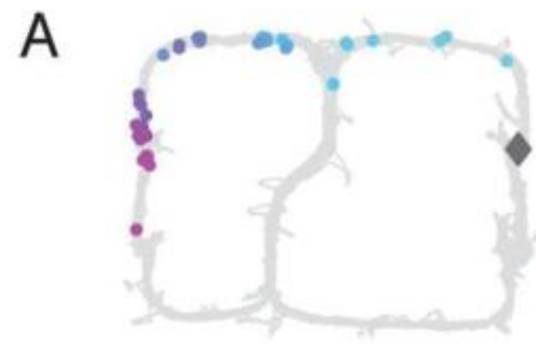
- http://rescience.github.io/

Forward replays:

Backward replays:

Shortcuts:





Gupta *et al*., 2010

Algorithm 3 Neural Dyna-Q with *prioritized sweeping* & multiple predecessors

INPUT: $\phi^{t=0}$, $\mathcal{N}_\mathcal{P}$, $\mathcal{G}_\mathcal{P}$, $\mathcal{N}_\mathcal{R}$, $\mathcal{G}_\mathcal{R}$
OUTPUT: $N_Q^{a\in\{N,S,E,W\}}$
PQueue $\leftarrow$ {} // PQueue: empty priority queue
nbTrials $\leftarrow$ 0
repeat
  a $\leftarrow$ softmax($N_Q(\phi^t)$)
  take action a, receive r, ϕ^{t+1}
  backprop(N_Q^a, $input = \phi^t$, $target = r + \gamma max_a(N_Q(\phi^{t+1}))$)
  Put ϕ^t in PQueue with priority $|N_Q^a(\phi^t) - (r + \gamma max_a(N_Q(\phi^{t+1})))|$
  if r> 0 **then**
    nbReplays $\leftarrow$ 0
    Pr = $\langle\rangle$ // empty list of predecessors
    repeat
      $\phi \leftarrow$ pop(PQueue)
      for each $G_P \in \mathcal{G}_P$ **do**
        if $G_P(\phi) > 0$ **then**
          $k \leftarrow$ index(G_P)
          append $N_P^k(\phi)$ to Pr
        end if
      end for
      for each $p \in$ Pr s.t norm(p) $> \epsilon$ **do**
        for each $a \in \{N, S, E, W\}$ **do**
          backprop(N_Q^a, $input$ = p, $target = N_R^a(p) + \gamma max_a(N_Q^a(\phi))$)
          Put p in PQueue with priority $|N_Q^a(p) - (N_R^a(p) + \gamma max_a(N_Q^a(\phi)))|$
          nbReplays $\leftarrow$ nbReplays + 1
        end for
      end for
    until PQueue empty **OR** nbReplays $\geq$ B
  end if
  $\phi^t \leftarrow \phi^{t+1}$
  nbTrials $\leftarrow$ nbTrials +1
until nbTrials = maxNbTrials

Algorithm 2 GALMO: Growing algorithm to learn multiple outputs

```
INPUT: 𝒮, 𝒩, 𝒢
OUTPUT: 𝒩, 𝒢
// 𝒮 = ⟨(in_0, out_0), ..., (in_n, out_n)⟩ : list of samples
// 𝒩 = ⟨N_0⟩ : lists of neural networks (outputs)
// 𝒢 = ⟨G_0⟩ : lists of neural networks (gates)
θ ← +∞
for nbepoch ∈ {1, maxepoch} do
  ℳ ← null // ℳ is a list of the minimal error per sample
  for each (in,out)∈ 𝒮 do
    ℰ ← null // ℰ is a list of errors for a sample
    for each N ∈ 𝒩 do
      append ‖N(in) − out‖_{L_1} to ℰ
    end for
    if min(ℰ) < θ then
      backprop(N_{argmin(ℰ)}, in, out)
      backprop(G_{argmin(ℰ)}, in, 1)
      for each G ∈ 𝒢 with G ≠ G_{argmin(ℰ)} do
        backprop(G, in, 0)
      end for
    else
      create N_new; append N_new to 𝒩
      N_new ← copy(N_{argmin(ℰ)})
      backprop(N_new, input =in, target =out)
      create G_new; append G_new to 𝒢
      backprop(G_new, in, 1)
    end if
  end for
  θ ← median(ℳ) + w ∗ (Q3(ℳ) − median(ℳ))
end for
```