Policy networks for Non-Markovian Deep RL Rewards

ANTONELLA ANGRISANI, ANDREA FANTI and LUCA LOBEFARO

TABLE OF CONTENTS

01

INTRODUCTION

Theoretical and experimental context

Temporal Goals with Reinforcement Learning

We can use Reinforcement Learning to solve Temporal Goals

However, temporal goals are **non-Markovian**

Extend the MDP with a DFA

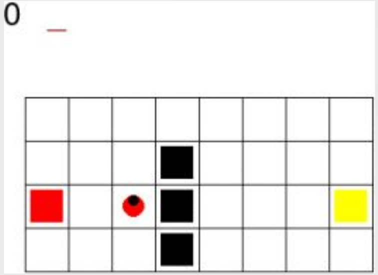
Simply appending DFA state and feeding to the agent network(s) does not work

Use |Q| separate networks (baseline)

Possible improvement for training time: share layers

Experiment Setting

SapientinoCase environment



- Low dimensional observation space: no need for CNNs
- Task can be simplified/made harder by simply changing the map

Advantage Actor-Critic

- Actor-Critic family: learn both policy and value separately
- On-policy
- Uses the Advantage as baseline

02

BACKGROUND

LTL_f / LDL_f for Non-Markovian Rewards

Reinforcement Learning and A2C

MARKOV ASSUMPTION

Action's effects depend only on the state in which it was executed, and a reward given at a state only on the previous action and state

NON-MARKOVIAN REWARDS

The reward depends on the state history rather than the last one only

WHY?

Create agents capable of learning to act so as to reach LTL_f / LDL_f goals

TEMPORAL SPECIFICATIONS

LTL_f / LDL_f formula can be converted to a DFA that recognize the same traces

$$\mathcal{M} = \langle S, A, Tr, R \rangle$$

where:

- S is a set of states, A a set of actions
- $Tr : S \times A \rightarrow Prob(S)$ a transition function that returns, for every state s and action a, a distribution over the next state
- R the reward function

MARKOV DECISION PROCESS

- $R : S \times A \rightarrow \mathbb{R}$ specifies the real-valued reward received by the agent when applying a in s
- the *policy* ρ assigns an action to each state, possibly conditioned on past states and actions
- the *value of policy* $v_\rho(s)$ is the expected sum of (discounted) rewards when starting at s and selecting actions based on ρ

NON-MARKOV-REWARD DECISION PROCESS

- $R : (S \times A)^* \rightarrow \mathbb{R}$ is a real-valued function over finite state-action sequences
- the *policy* ρ induces a distribution over the set of possible infinite traces
- the *value of policy* $v_\rho(s)$ is the expected value of infinite traces, where the distribution over traces is defined by the initial state s_0, the transition function Tr, and the policy ρ

$\mathbf{LTL_f}$ / $\mathbf{LDL_f}$

Linear Temporal Logic (LTL) over finite traces

Formulas are built from a set $\mathcal{P}$ of propositional symbols and are closed under the boolean connectives, the unary temporal operator $\circ$ (*next-time*) and the binary temporal operator $\mathcal{U}$ (*until*):

$$\varphi ::= A \mid (\neg\varphi) \mid (\varphi_1 \wedge \varphi_2) \mid (\circ\, \varphi) \mid (\varphi_1 \mathcal{U} \varphi_2)$$

Linear Dynamic Logic (LDL) over finite traces

LDL_f is an extension of LTL_f: the latter is merged with Regular Expressions (RE):

$$\varphi ::= tt \mid \neg\, \varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\varrho\rangle\varphi$$
$$\varrho ::= \phi \mid \varphi? \mid \varrho_1 + \varrho_2 \mid \varrho_1 ; \varrho_2 \mid \varrho^*$$

Deterministic Finite Automata (DFA)

Each LDL_f formula φ can be associated with a Nondeterministic Finite Automata $A_\varphi = \langle\Sigma, Q, q_0, \delta, F\rangle$*, if the NFA accepts exactly the traces satisfying the formula.
The NFA can be transformed into a DFA on-the-fly (avoiding the entire construction of A_φ):

- progress all possible states that the NFA can be in
- accept the trace iff the set of possible states contains a final state.

* Σ is a finite nonempty alphabet; Q is a finite nonempty set of states; $q_0 \in Q$ is the initial states; $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation; $F \subseteq Q$ is the set of final states

LDL_f advantages

01 ENHANCED EXPRESSIVE POWER

greater expressivity, and possibility to represent also procedural constraints (sequencing constraints)

02 MINIMALITY AND COMPOSITIONALITY

if the current MDP was minimal, the extended MDP is minimal too: it is enough that each DFA A_φ is minimal

if a new formula is added, we need simply to extend MDP with one additional component.

03 FORWARD CONSTRUCTION VIA PROGRESSION

to ensure, on the base of the initial state, the generation of reachable states only

Non-Markovian-Rewards

LDL_f provides an intuitive language for specifying R, using a set of pairs $\{\varphi_i, r_i\}_{i=1}^m$.

If the current (partial) trace is $\pi = \langle s_0, a_1, \ldots, s_{n-1}, a_n \rangle$, the agent receives at s_n a reward r_i for every formula φ_i satisfied by π:

$$R(\pi) = \sum_{1 \le i \le m : \pi \models \varphi_i}^{n} r_i$$

Theorem: The NMRDP $\mathcal{M} = \langle S, A, Tr, \{\varphi_i, r_i\}_{i=1}^m \rangle$ is equivalent to the extended *MPD* $\mathcal{M}' = \langle S', A', Tr', R' \rangle$ defined as:

$$S' = Q_1 \times \cdots \times Q_m \times S$$

$$Tr' : S' \times A \times S' \to [0,1] \qquad Tr'(q_1, \ldots, q_m, s, a, q_1', \ldots, q_m', s') = \begin{cases} Tr(s, a, s') & if\ \forall i:\ \delta_{i(q_i, s)} = q_i' \\ 0 & otherwise \end{cases}$$

$$R' : S' \times A \times S' \to \mathbb{R} \qquad R'(q_1, \ldots, q_m, s, a, q_1', \ldots, q_m', s') = \sum_{i : q_i' \in F_i} r_i$$

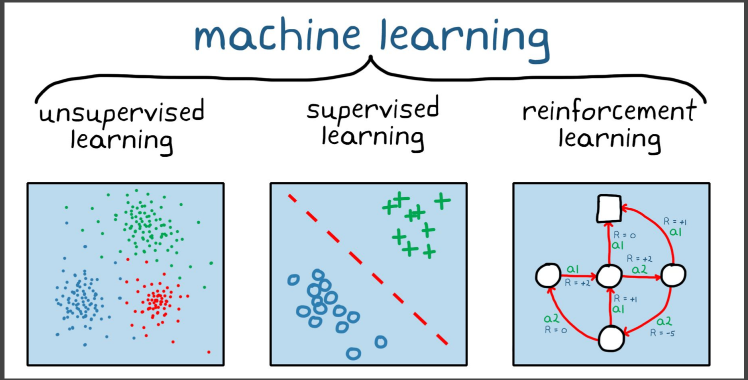
Lemma: Given an NMRDP $\mathcal{M}$ and an equivalent MDP $\mathcal{M}'$, every policy ρ' for $\mathcal{M}'$ has an equivalent policy ρ for $\mathcal{M}$ and viceversa.

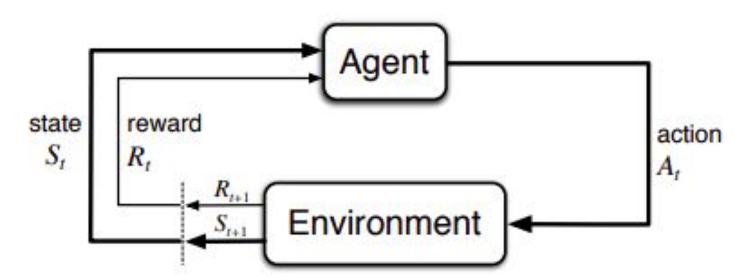
RL for NMRDP with LTL_f LDL_f rewards

Learn a (possibly optimal) policy for an NMRDP $\mathcal{M} = \langle S, A, Tr, \{\varphi_i, r_i\}_{i=1}^{m} \rangle$, whose rewards r_i are offered on traces specified by LTL_f / LDL_f formulas φ_i and where the LTL_f / LDL_f reward formulas $\{\varphi_i, r_i\}_{i=1}^{m}$ and the transitions function Tr is hidden to the learning agent.

Theorem: RL for the LTL_f / LDL_f rewards over an NMRDP $\mathcal{M} = \langle S, A, Tr, \{\varphi_i, r_i\}_{i=1}^{m} \rangle$, with Tr and $\{\varphi_i, r_i\}_{i=1}^{m}$ hidden to the learning agent can be reduced to RL over the MDP $\mathcal{M}' = \langle S', A', Tr', R' \rangle$, with Tr' and R' hidden to the learning agent .

What is Reinforcement Learning?





Markov properties:

$$P(S_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t, ..., S_0, A_0)$$
$$P(R_t|S_t, A_t) = P(R_t|S_t, A_t, ..., S_0, A_0)$$

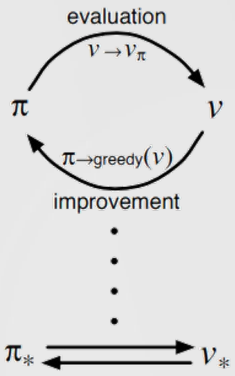
How an agent can learn from experience

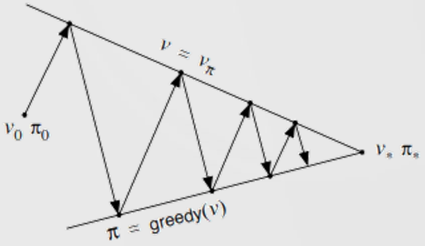
- It has to learn an optimal policy
- An optimal policy maximize the expected return
- V-value function

$$V^{\pi}(s) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi]$$

- Q-value function

$$Q^{\pi}(s, a) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi]$$





The optimal policy is computed as:

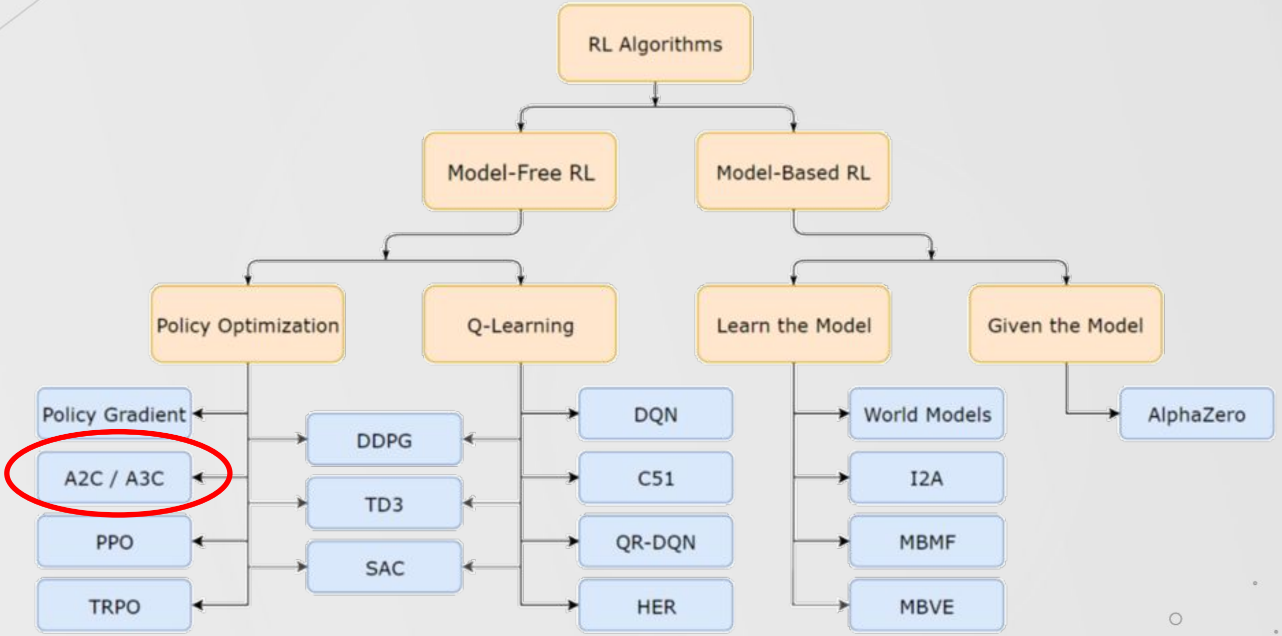
$$\pi^*(s) = \max_{a \epsilon A} Q^*(s, a)$$

where:

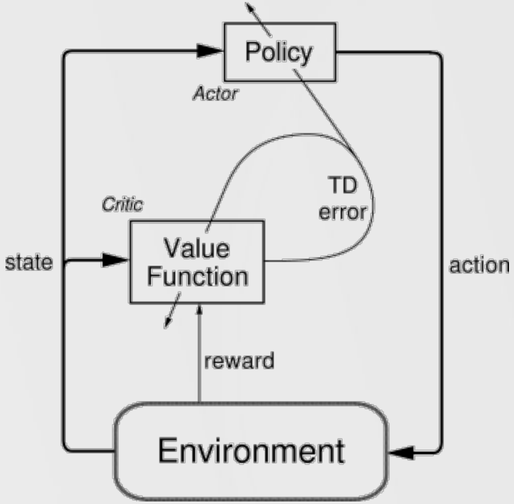
$$V^*(s) = \max_{\pi \epsilon \Pi} V^{\pi}(s)$$

$$Q^*(s, a) = \max_{\pi \epsilon \Pi} Q^{\pi}(s, a)$$

Types of RL algorithms



Actor-Critic



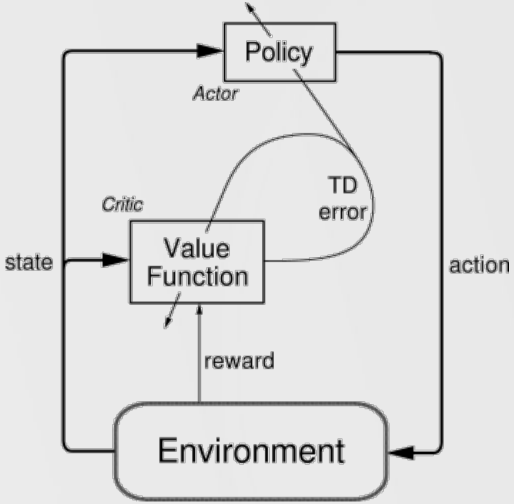
- The **Actor** network approximates the policy function
- The **Critic** network approximates the value function

- Policy gradient:

$$\nabla_\theta J(\theta) = E_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)G_t\right]$$

- Problems of this formulation:
  - high variability in log probabilities
  - noisy gradients
  - unstable learning

Advantage Actor-Critic (A2C)



- To solve unstability a baseline is introduced, the **Advantage**:

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t) A(s_t, a_t) = r_{t+1} + \gamma V_v(S_{t+1}) - V_v(s_t)$$

(it tells about the extra reward that could be obtained by the agent by taking that particular action and it is computed using the TD Error estimator that is the difference between consecutive temporal predictions)

- Then we obtain:

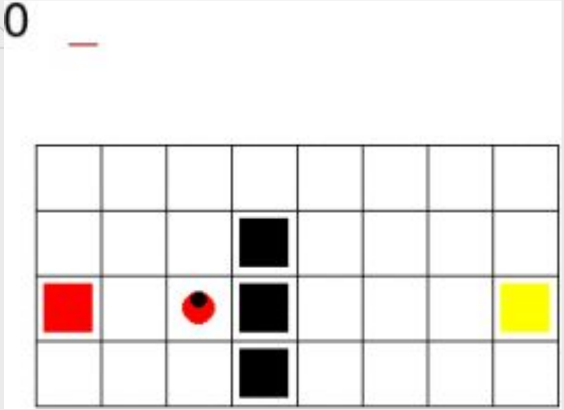
$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(S_{t+1}) - V_v(s_t))$$

03

AGENT AND ENVIRONMENT

SapientinoCase and Agent models

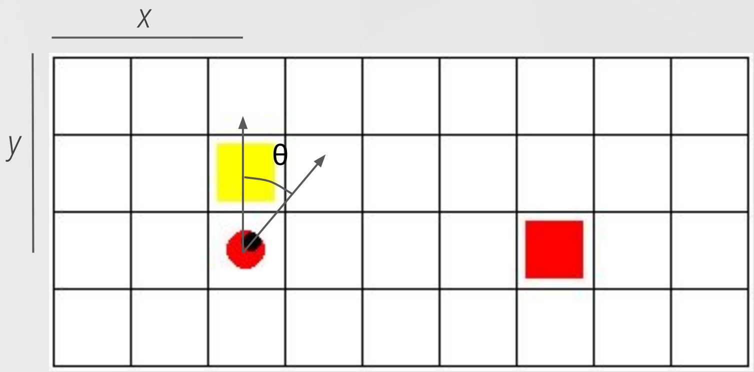
SapientinoCase



- A unicycle robot navigates a 2d grid
- Cells can be empty, full, colored
- (non-Markovian) goal is to “visit” colored cells in a specific order

Observations

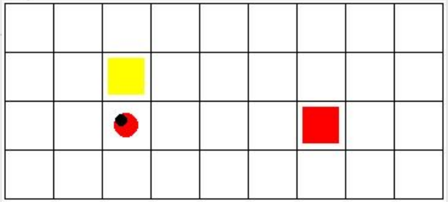
Continuous in the form $s = (x, y, \sin\theta, \cos\theta)$



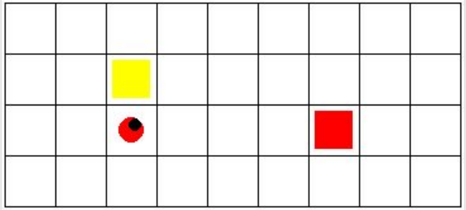
Actions

Discrete in 0..4

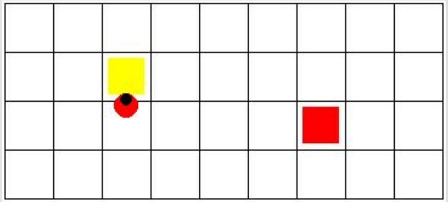
0 (turn left)



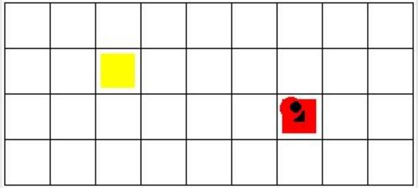
1 (turn right)



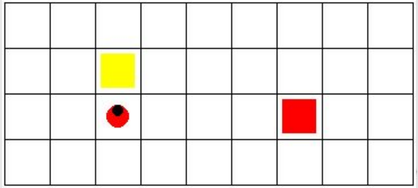
2 (forward)



3 (*beep*/visit)



4 (no-op)



Non-Markovian Goal

The reward depends on previous transitions → non-Markovian

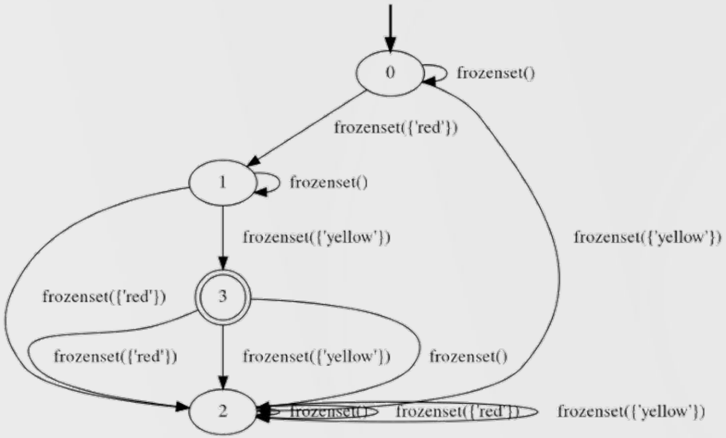
Use the approach from “$LTLf/LDLf$ Non-Markovian Rewards” (Brafman, De Giacomo, Patrizi 2018):

1. Use a temporal formula to model the reward
2. Compute the associated DFA
3. Augment the base environment state with the DFA state and reward

This makes it possible to integrate non-Markovian goals in an MDP by producing an extended MDP

Generated DFA

Example DFA for the 2 colors case



In our simple case, the DFA was generated manually (not automatically from a temporal formula)

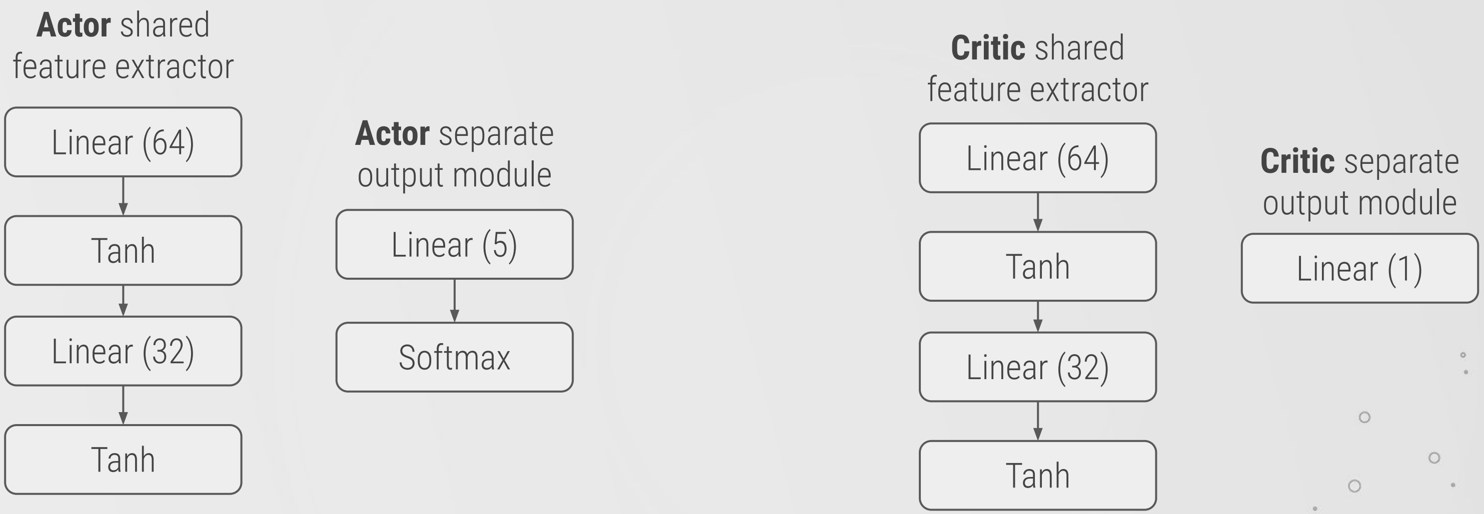
- N + 2 states (for each color + initial and sink)
- Transition to next color when on the correct cell and last action was visit
- Transition to sink whenever wrong color is visited

Actor model(s): Baseline

Baseline approach: N identical separate networks, one for each automaton state

Possible Improvement

Share layers (up to the second last)



Implementation (PyTorch)

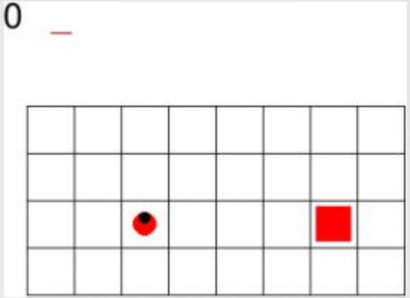
- Subclass Module and use ModuleList to store all of the networks
- For the shared configuration, have a common network and separated output layers
- The correct network/output layer is selected in the forward method by indexing the ModuleList with the automaton state

04

EXPERIMENTS

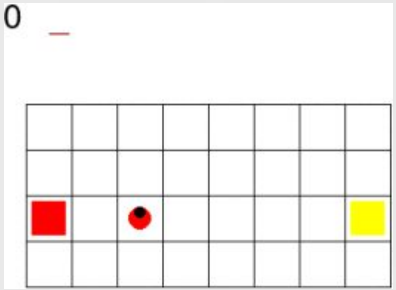
Experiments performed and results obtained

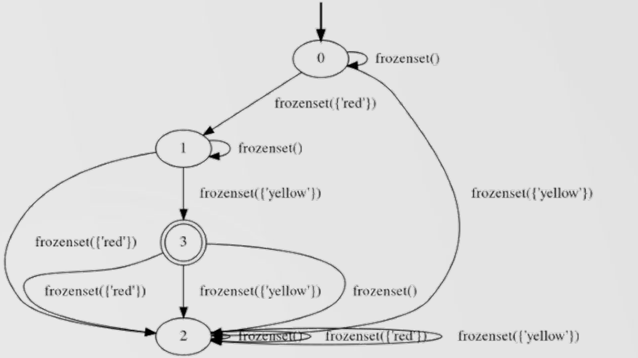
Simpler case: 1 color



PARAMETERS
max time-steps: 300
learning rate: 1e-3
gamma: 0.99
episodes: 2700

Non-Markovian Setting: 2 colors





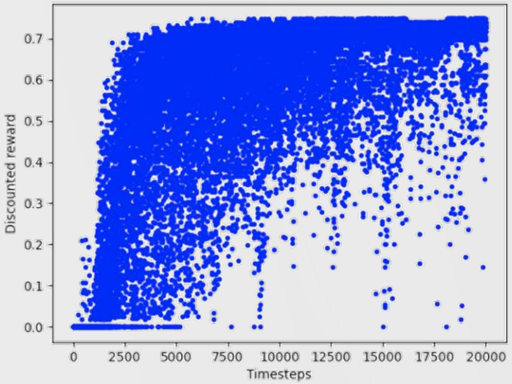


Fig. 7. Evolution of the discounted rewards during training for the two colors case with separated networks

PARAMETERS
max time-steps: 400
learning rate: 5e-4
gamma: 0.99

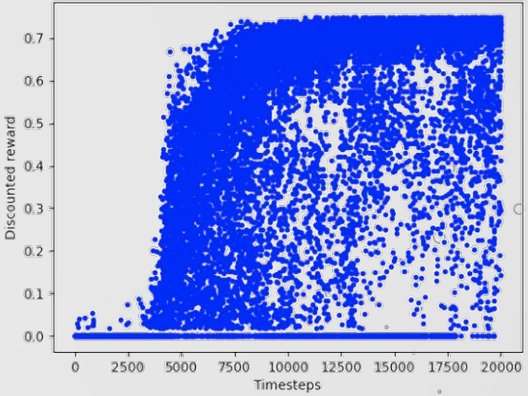
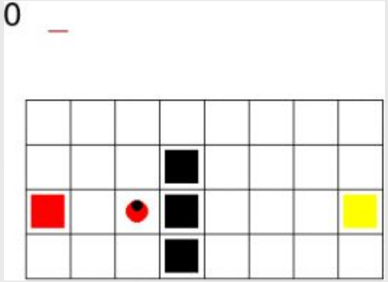
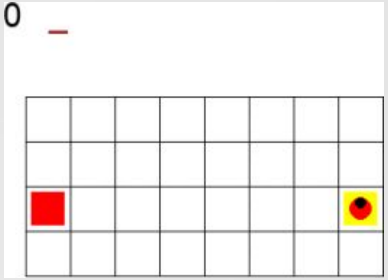
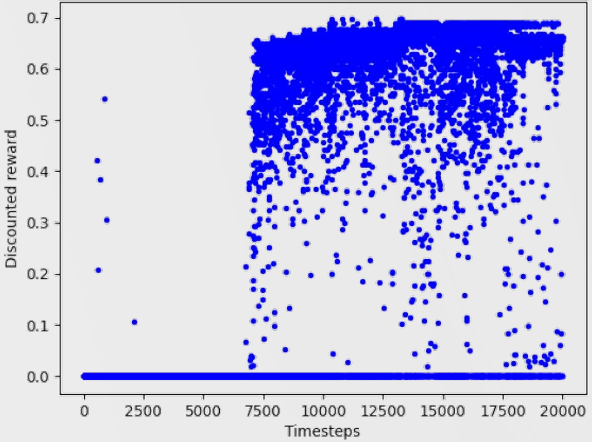


Fig. 8. Evolution of the discounted rewards during training for the two colors case with unified networks

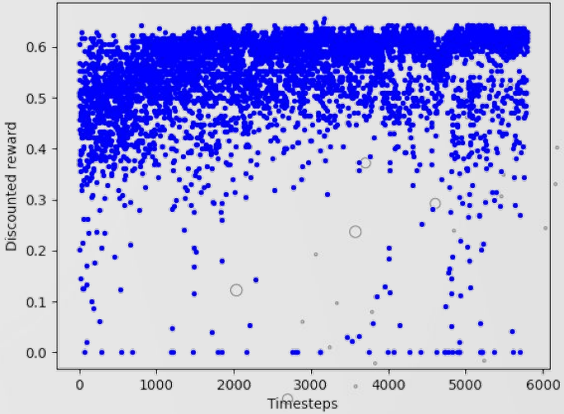
Transfer Learning



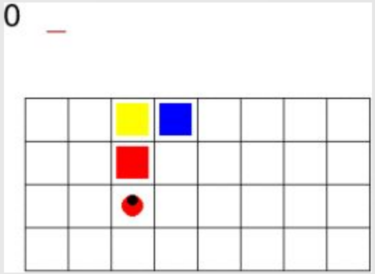


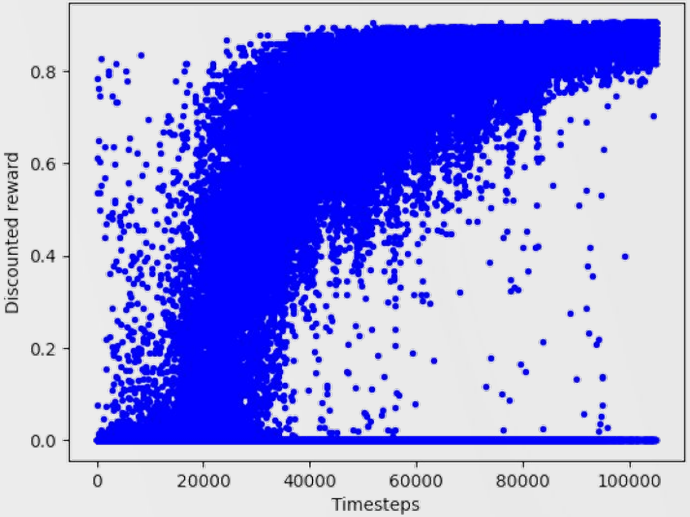


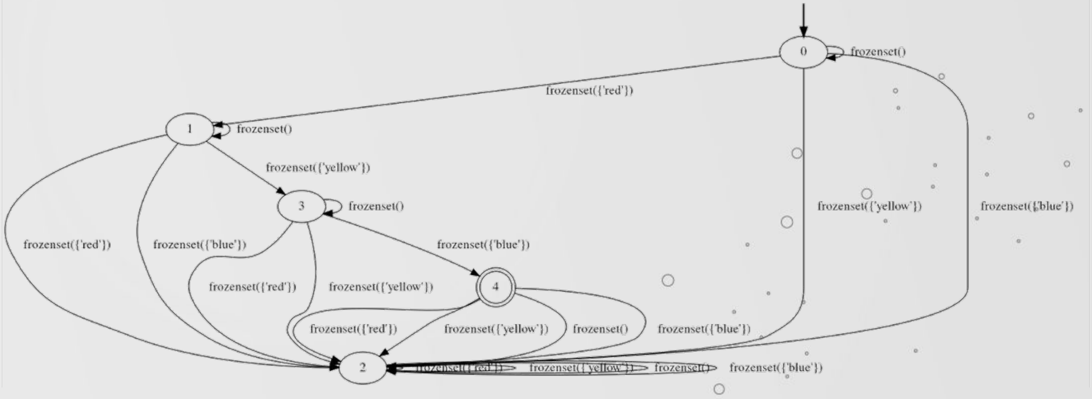
PARAMETERS
max time-steps: 400
learning rate: 5e-4
gamma: 0.99



3 colors setting







05

CONCLUSIONS

Conclusions and Future Works

Conclusions

- A shared neural architecture is able to accomplish a non-Markovian task in the same amount of time of the baseline but with less memory
- With a pre-trained model we can solve harder configurations in a sensible less amount of episodes

EXTENSIONS

- A three color map with an harder configuration
- More than two colors with reward shaping
- A more complex environment (e.g. Atari gym environments)

Thanks for the Attention