Policy networks for Non-Markovian Deep RL

Reasoning Agent project

Roberto Cipollone

cipollone@diag.uniroma1.it

PhD student

Sections

Markov assumptions

RL with temporal specifications

Deep RL networks

Temporal goals

General objective

Create agents capable of reaching LTL_f goals.

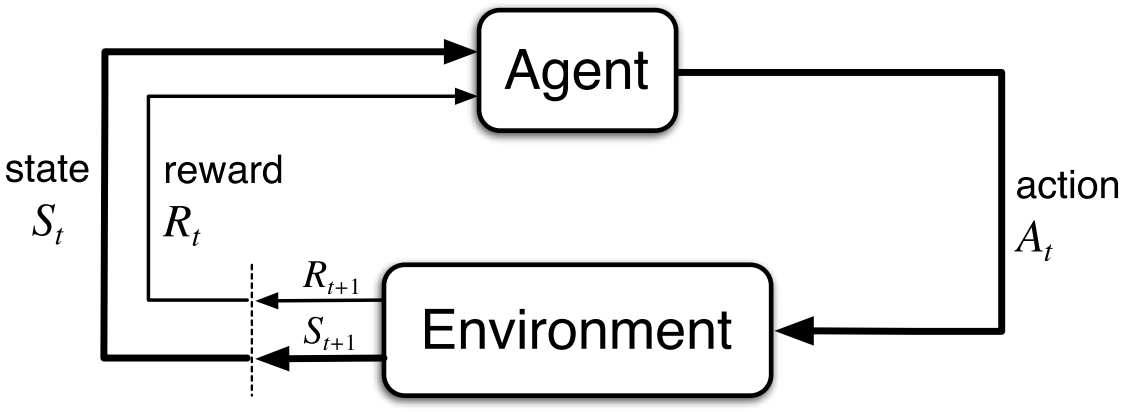
LTL_f: $\Box\neg \mathit{Falling} \wedge \Diamond \mathit{AtDoor}$
LDL_f: $\langle\, \mathit{true}^* \,\rangle(\mathit{AtDoor} \wedge \langle\, \mathit{true}^* \,\rangle \mathit{AtDesk})$

A good execution is $\pi \models \varphi$.

Solution method: we'll do Reinforcement Learning (RL).

Reinforcement Learning

The general setting of Reinforcement Learning[1]



Most RL algorithms assume the environment can be modelled with a *Markov Decision Process*.

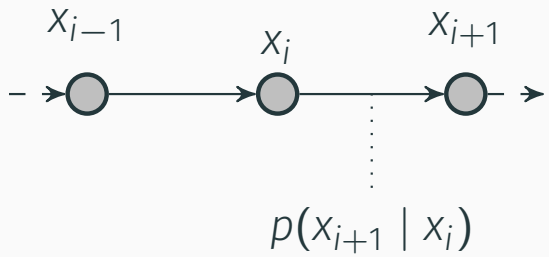
[1] Sutton and Barto, *Reinforcement learning: an introduction*.

Markov assumptions

Bayes nets

A representation of probabilities with Directed Acyclic Graphs.

A Markov chain:



A missing arc means that two variables are conditionally independent:

$$x_{i+1} \perp x_{i-1} \mid x_i$$

Gray nodes ● are visible quantities; ○ are hidden quantities.

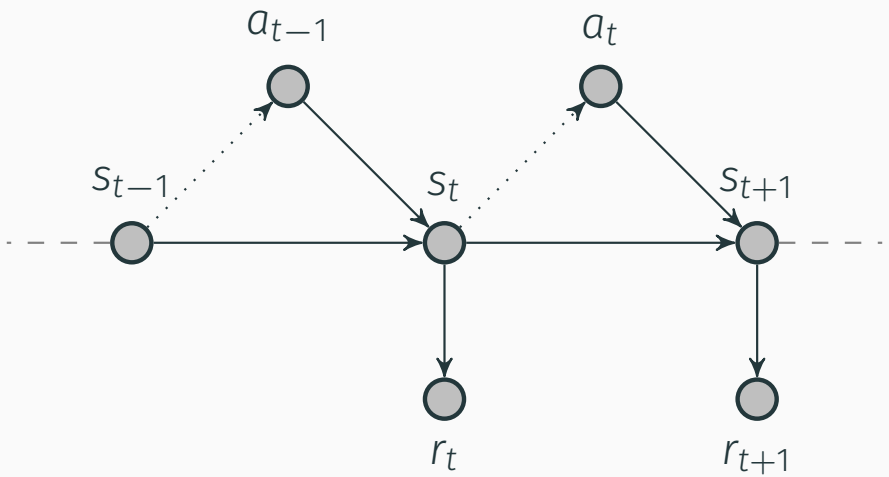
Markov Decision Process

Definition

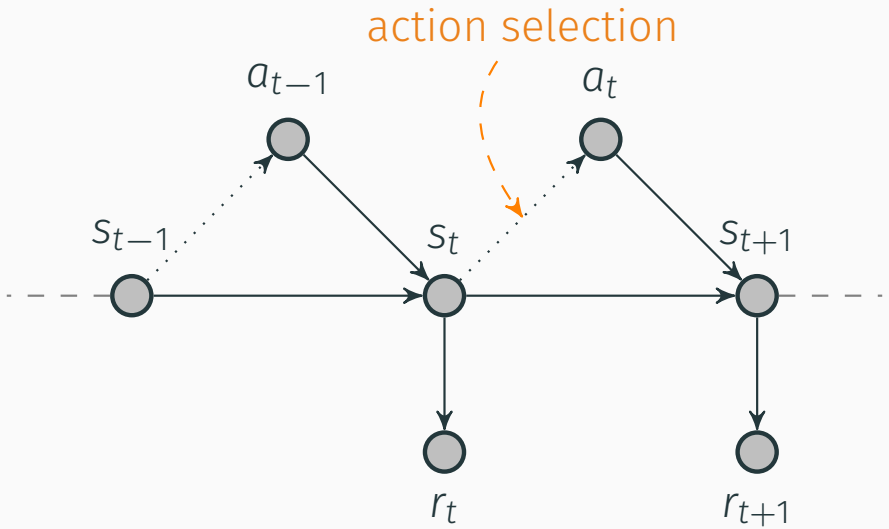
A Markov Decision Process (MDP) is a tuple $\langle S, A, T, R, \gamma \rangle$, where:

- States S, actions A, discount factor γ;
- Transition function $T(s, a, s') = p(s' \mid s, a)$;
- Reward function $R(s, a, s') \in \mathbb{R}$.

Most RL algorithms assume an MDP model of the world.



Markov assumptions



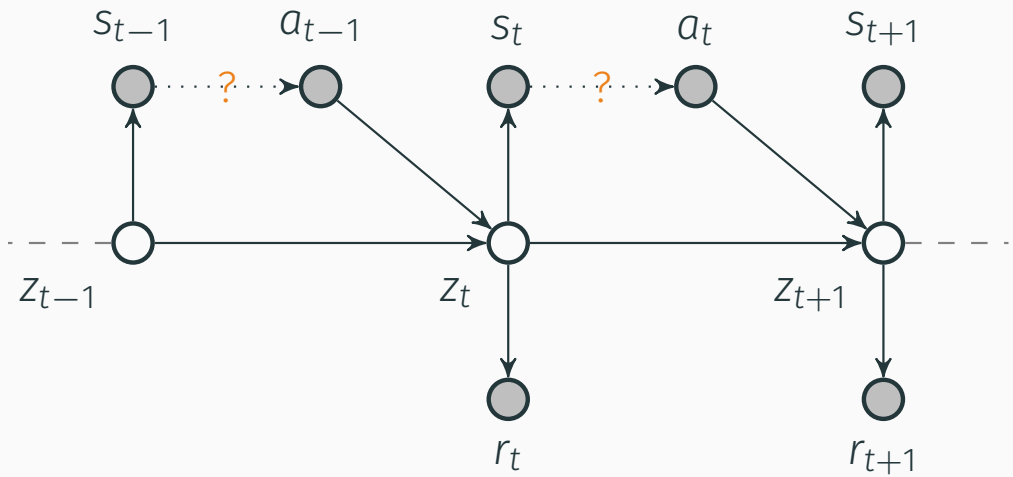
Properties of an MDP:

$$s_{t+1} \perp s_0, \ldots, s_{t-1} \mid s_t \qquad \text{for all } t$$

$$r_{t+1} \perp s_0, \ldots, s_{t-1} \mid s_t \qquad \text{for all } t$$

About failures on POMDPs

A Partially Observable MDP:



Do the properties still hold? No!

$$s_{i+1} \not\perp s_0, \ldots, s_{i-1} \mid s_i \qquad \text{for some } i$$

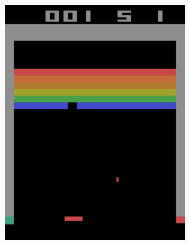
$$r_{i+1} \not\perp s_0, \ldots, s_{i-1} \mid s_i \qquad \text{for some } i$$

We can't just learn a policy $\rho : S \to A$.

About failures on POMDPs

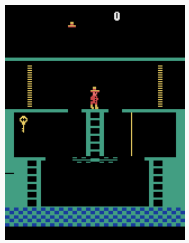
What if I do it anyway? “We have Networks after all”

Breakout



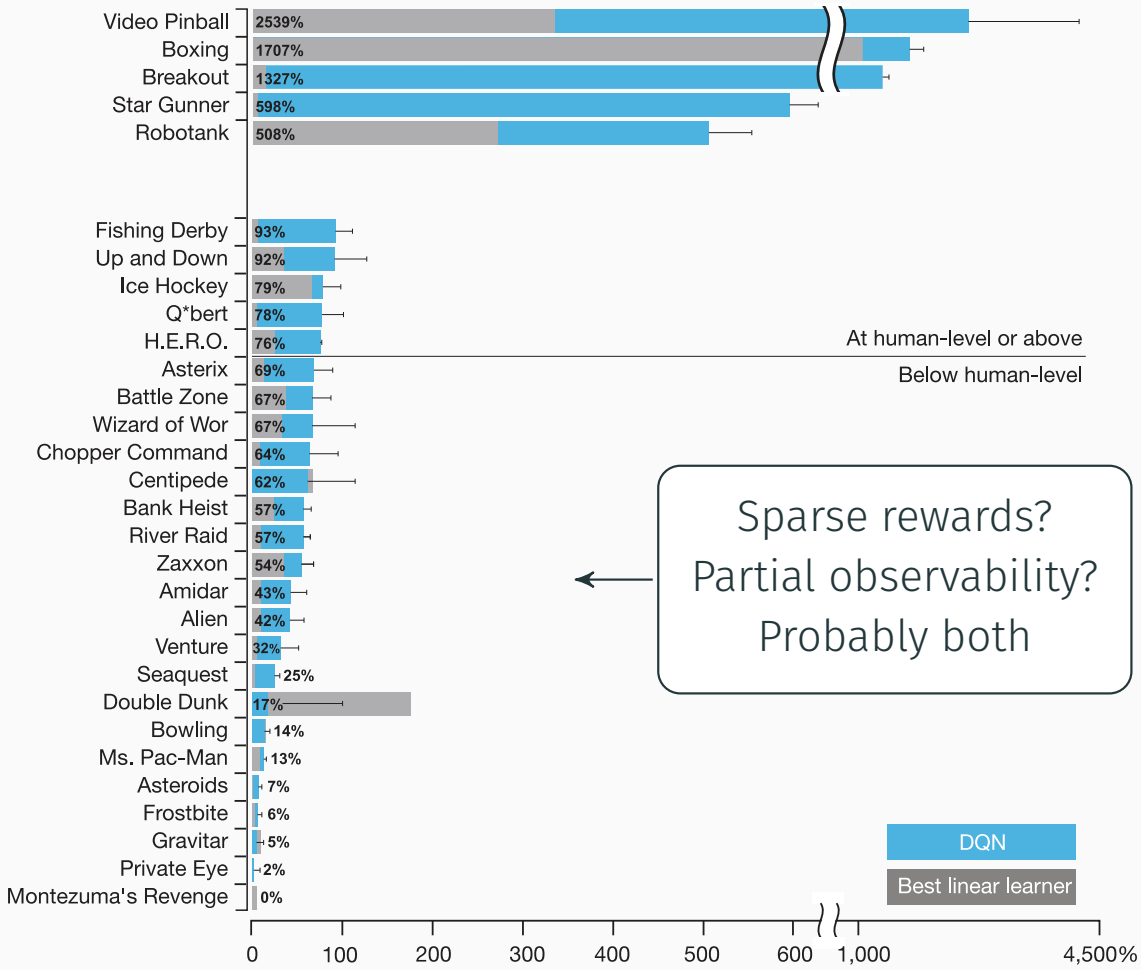
- Did I just hit the ball?
- What is the direction of the ball?
- Simple workaround: remember 4 frames.

Montezuma’s Revenge



- Did I open the door in room number 10?
- Simple workaround: *not exactly*.

Difficult environments



Mnih et al., “Human-level control through deep reinforcement learning”

A specific class of non-Markovian environments

The general POMDP class cannot be solved.

Best to adopt a more restrict model of the environment:
see Alessandro Ronca's presentation.

Here, instead, we'll only look at temporal specifications.

RL with temporal specifications

Temporal logics for reward specification

Assume we're given an MDP dynamics: $\langle S, A, T \rangle$.

How to reward *behaviours*

We want to define a non-Markovian reward function:

$$\bar{R} : (S \times A)^* \to \mathbb{R}$$
$$\bar{R} : \langle s_1, a_1, \ldots, a_{t-1}, s_t \rangle \mapsto r_t$$

We know temporal logics have been designed to talk about *desirable executions*...

RL with temporal specifications

Steps:

- Define an **alphabet** of propositional symbols, $\mathcal{F}$ (fluents).
- Define a labelling function $f_{\mathcal{F}} : S \times A \to 2^{\mathcal{F}}$ (grounding).

These definitions induce a trace π of propositional interpretations:

$$(S \times A)^* \xrightarrow{f_{\mathcal{F}}} (2^{\mathcal{F}})^*$$

We can now define a set $\{(\varphi^{(i)}, r^{(i)})\}_i$ to specify non-Markovian rewards[2]:

$$\bar{R}(\pi) := \sum_{i:\pi \models \varphi^{(i)}} r^{(i)}$$

[2] Brafman, De Giacomo, and Patrizi, "LTLf / LDLf non-markovian rewards".

Rewarding with automata

Any LTL_f / LDL_f formula can be converted to a DFA that recognizes the same traces: denote with $\mathcal{A}_\varphi = \langle Q_\varphi, q_{0_\varphi}, \Sigma, F_\varphi \rangle$ the DFA for φ.

LTL_f / LDL_f rewards

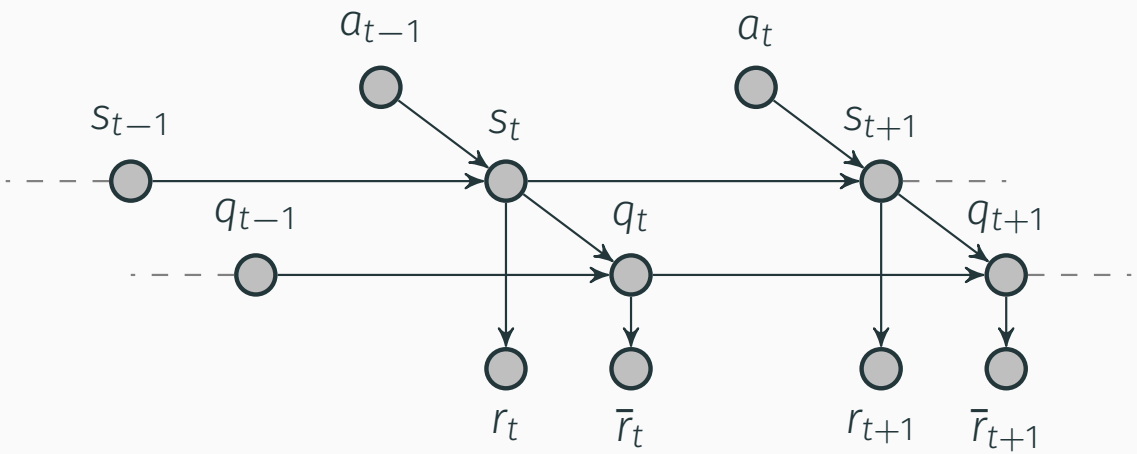
A Non-Markovian Reward Decision Process (NMRDP) $\langle S, A, T, \bar{R} \rangle$, with $\bar{R}$ defined by $\{(\varphi^{(i)}, r^{(i)})\}_{i=1}^{m}$, is *equivalent* to an MDP with state space:

$$S' := S \times Q_{\varphi^{(1)}} \times \cdots \times Q_{\varphi^{(m)}}$$

Restraining Bolt formulation

- An initial MDP is given $\langle S, A, T, R \rangle$.
- We define an additional specification $\langle \mathcal{F}, \{(\varphi^{(i)}, r^{(i)})\}_{i=1}^{m} \rangle$, with associated labelling function $f_{\mathcal{F}}$. Then, there exists an equivalent MDP with states S' and rewards $R + \bar{R}$.

One DFA state is enough



Without restrictions, we can always define an equivalent MDP with state state $S \times Q$:

- By synchronization of the DFAs[3].
- By minimization[4].

[3] De Giacomo, Iocchi, et al., "Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications".

[4] De Giacomo, Favorito, et al., "Temporal Logic Monitoring Rewards via Transducers".

Deep RL networks

Deep RL algorithms

Deep RL is useful when $|S|$ is very large.

Deep RL

The main idea is to use a Neural Network to estimate functions on S. Which functions?

- State values $V(s) \approx f_{\text{net}}(s; \theta)$: **value** methods;
- Policies $\rho(s) \approx f_{\text{net}}(s; \theta)$: **policy gradient** methods;
- Combination of both.

We'll look at DQN variants[5].

[5] Mnih et al., "Human-level control through deep reinforcement learning".

DQN

As in classic Q-learning, for a transition $t_i = (s, a, r, s')$, we define targets:

$$Y_i := r + \gamma \max_{a' \in A} \hat{Q}(s', a')$$

and update Q to minimize the distance:

$$l_i = (\hat{Q}(s, a) - Y_i)^2$$

DQN approximates Q with a network: $Q^*(s, a) \approx \hat{Q}(s, a; \theta)$.

Careful with function approximators

Pointwise updates can have a global impact.

DQN

Techniques for improved stability

Experience replay For learning, we sample transitions from a replay buffer.

Target network For targets, Y_i, we use a network which changes more rarely.

And other minor choices which are specific to Atari games.

Many extensions since then: Double DQN, Dueling DQN, Distributional DQN, ...[6].

Not all must be DQN!

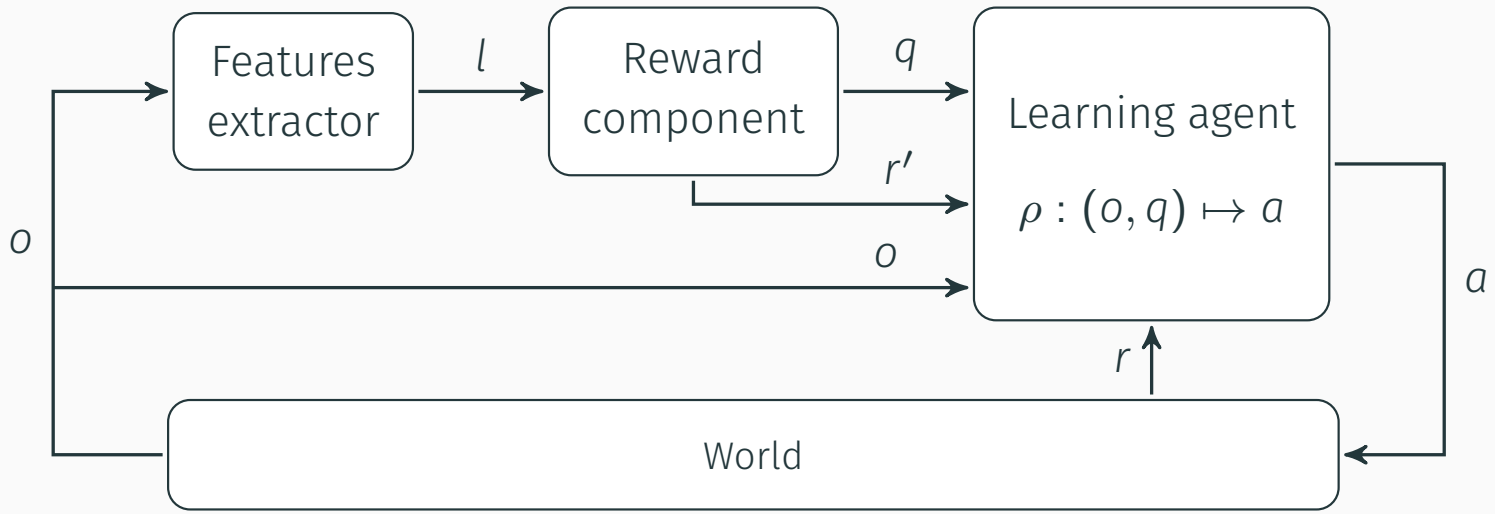
[6] François-Lavet et al., "An Introduction to Deep Reinforcement Learning".

And supervised learning

In addition, everything about supervised learning applies:

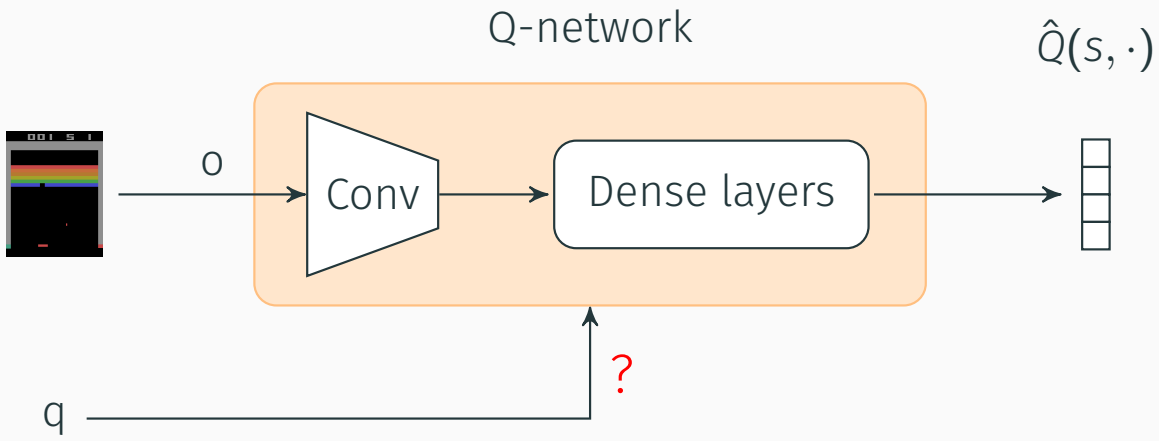
- Convergence to local optimum and initialization.
- Representativeness of the dataset.
- Underfitting, overfitting.

DQN agent for non-Markovian rewards



We need to choose the structure of ρ.

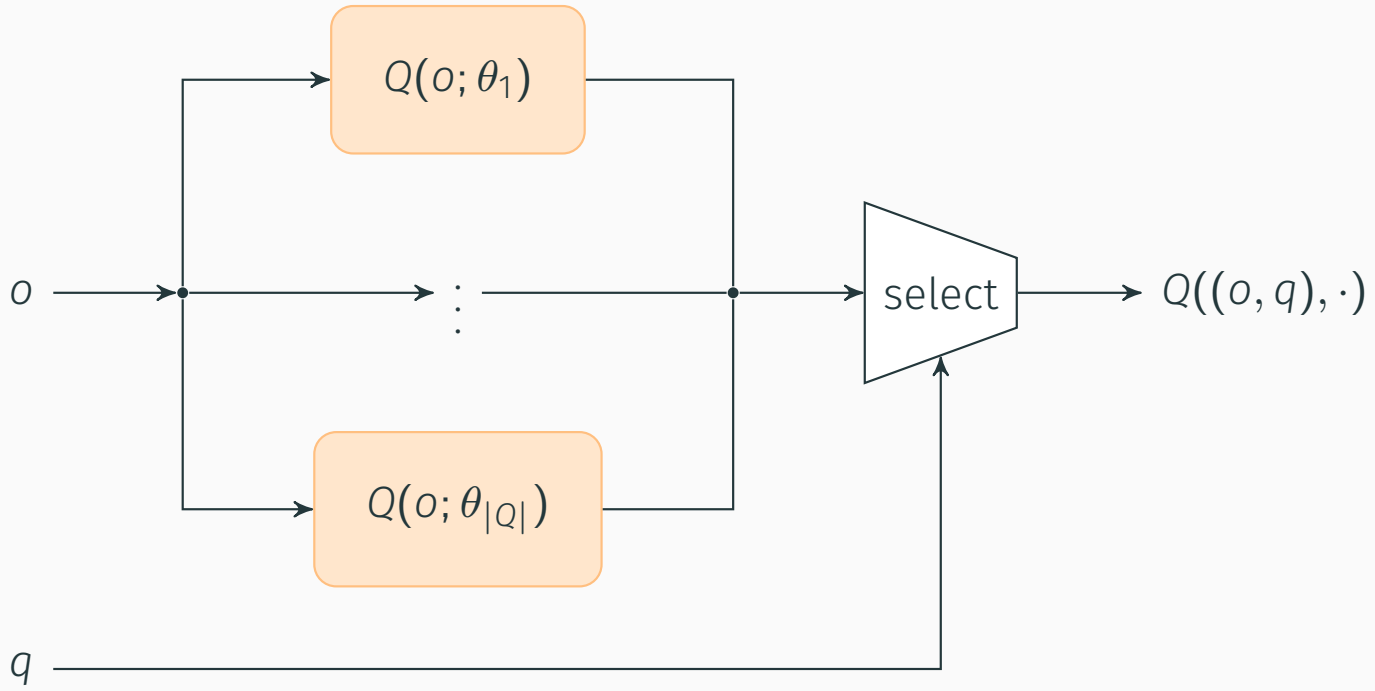
We need to design an appropriate network



The network needs to be re-designed

The baseline implementation

As baseline, we'll consider this implementation:



A number of $|Q|$ separate "experts".

The baseline implementation

Discussion:

- States have maximum importance.
- High learning stability.

Downsides:

- Memory usage increases linearly with $|Q|$.
- Training time and samples needed increase at least with $|Q|$.

Possible improvements

Many possible improvements:

Improvements on the model

Let the networks share part of the weights.

Motivation:

- The first convolutional layers learn input encodings.
- Most common knowledge about the observations is shared among subtasks.

Possible improvements

Improvements on the algorithm

Prioritized experience replay

Motivation:

- The structure of $s = (o, q)$, can be exploited to give a bias to sampling.
- Changing DFA state is a significant change.
- Might improve training stability (to be verified).

Projects: what to do

Independent of your choice:

- Study about LTL_f/LDL_f[7].
- Study about RL under LTL_f/LDL_f goals[8].
- Understand the baseline implementation I'll provide.

[7] De Giacomo and Vardi, "Linear temporal logic and Linear Dynamic Logic on finite traces".

[8] De Giacomo, Iocchi, et al., "Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications".

Projects: what to do

Of your choice:

- Choose and study a Deep RL algorithm[9].
- Select the implementation of your choice. I suggest among:
  1. TensorForce
  2. Stable-baselines
  3. TF-agents
  4. Keras-RL
- Select any Gym environment.
- Write your own temporal goals.
- Experiment with the baseline network.
- Propose and *motivate* an improvement from the baseline.

[9] Start from François-Lavet et al., "An Introduction to Deep Reinforcement Learning"

Policy networks for Non-Markovian Deep RL

Reasoning Agent project

Roberto Cipollone

cipollone@diag.uniroma1.it

PhD student

References

- Brafman, Ronen I., Giuseppe De Giacomo, and Fabio Patrizi. “LTLf / LDLf non-markovian rewards”. In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (2018).
- De Giacomo, Giuseppe, Marco Favorito, et al. “Temporal Logic Monitoring Rewards via Transducers”. In: July 2020.
- De Giacomo, Giuseppe, Luca Iocchi, et al. “Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications”. In: *Proceedings International Conference on Automated Planning and Scheduling, ICAPS* (2019).
- De Giacomo, Giuseppe and Moshe Y. Vardi. “Linear temporal logic and Linear Dynamic Logic on finite traces”. In: *IJCAI International Joint Conference on Artificial Intelligence* (2013).
- François-Lavet, Vincent et al. “An Introduction to Deep Reinforcement Learning”. In: *Foundations and Trends in Machine Learning* (2018).
- Mnih, Volodymyr et al. “Human-level control through deep reinforcement learning”. In: *Nature* (2015).
- Sutton, Richard S. and Andrew G. Barto. *Reinforcement learning: an introduction*. 2018.