Multi-Advisor Reinforcement Learning

Romain Laroche[*]
Microsoft Maluuba
Montréal, Canada

Mehdi Fatemi
Microsoft Maluuba
Montréal, Canada

Joshua Romoff
Microsoft Maluuba
McGill University
Montréal, Canada

Harm van Seijen
Microsoft Maluuba
Montréal, Canada

Abstract

This article deals with a novel branch of Separation of Concerns, called Multi-Advisor Reinforcement Learning (MAd-RL), where a single-agent RL problem is distributed to n learners, called advisors. Each advisor tries to solve the problem with a different focus. Their advice is then communicated to an aggregator, which is in control of the system. For the local training, three off-policy bootstrapping methods are proposed and analysed: *local-max* bootstraps with the local greedy action, *rand-policy* bootstraps with respect to the random policy, and *agg-policy* bootstraps with respect to the aggregator's greedy policy. MAd-RL is positioned as a generalisation of Reinforcement Learning with Ensemble methods. An experiment is held on a simplified version of the Ms. Pac-Man Atari game. The results confirm the theoretical relative strengths and weaknesses of each method.

1 Introduction

When a person faces a complex and important problem, his individual problem solving abilities might not suffice. He has to actively seek for advice around him: he might consult his relatives, browse different sources on the internet, and/or hire one or several people that are specialised in some aspects of the problem. He then aggregates the technical, ethical and emotional advice in order to build an informed plan and to hopefully make the best possible decision. In this article, we propose to model this conduct under a novel multi-agent framework, which we call Multi-Advisor Reinforcement Learning (MAd-RL).

Formalised in Section 2, MAd-RL intends to partition a single-agent Reinforcement Learning [RL, Sutton and Barto, 1998] into a Multi-Agent RL problem [Shoham *et al.*, 2003], under the widespread *divide & conquer* paradigm. Unlike Hierarchical RL [Dayan and Hinton, 1993; Parr and Russell, 1998; Dietterich, 2000b], our approach places all agents at the same level and gives them the role of advisors. This role consists in providing an aggregator with the local Q-values for each available action. We position MAd-RL as a generalisation of RL with Ensemble methods [Dietterich, 2000a], allowing both the fusion of several weak RL learners, and the decomposition of a single-agent RL problem into concurrent subtasks, by allocating advisors to focus on different aspects of the problem such as reward channels. Only a few papers [Wiering and Van Hasselt, 2008; Harutyunyan *et al.*, 2015; van Seijen *et al.*, 2017] tackled the confluence of RL and Ensemble methods. All of them follow a method where agents are trained independently and greedily to their local optimality, and are aggregated into a global policy by voting or averaging. This paper is also the first to theoretically analyse the local greedy bootstrapping method.

Section 3 shows that this method, which we call *local-max*, presents the severe theoretical shortcoming of inverting a $\max\sum$ into a $\sum\max$ into the global Bellman equation. In practice, this $\max\sum$ inversion causes some states to become *attractors*: an attractor is a state where advisors are pulling in different directions equally and where the *local-max* aggregator's solution is to remain static. Two novel, attractor-free, off-policy bootstrapping methods are proposed and analysed: *rand-policy* bootstrapping method guarantees the convergence to a fair short-sighted policy, but also has the drawback to prevent efficient long-term planning; and *agg-policy* bootstrapping method optimises the system with respect to the global optimal Bellman equation, but without any convergence guarantee in the general case.

van Seijen *et al.* [2017] showed on a simplified version of Ms Pac-Man, called Pac-Boy, that a MAd-RL architecture with *local-max* significantly speeds up learning and converges to a better solution than several deep RL baselines. Section 4 extends this effort and provides empirical insights for MAd-RL by comparing the bootstrapping methods and confirm their theoretical analyses: *local-max* gets very unstable as soon as some noise is introduced, whereas *agg-policy* achieves similar scores with robustness to noise.

[*] romain.laroche@microsoft.com

2 Multi-Advisor Reinforcement Learning

2.1 Markov Decision Process

The Reinforcement Learning (RL) framework is formalised as a Markov Decision Process (MDP). An MDP is a tuple $\langle \mathcal{X}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{X}$ is the state space, $\mathcal{A}$ is the action space, $P : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$ is the Markovian transition stochastic function, $R : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ is the immediate reward stochastic function, and γ is the discount factor.

A *trajectory* $\langle x(t), a(t), x(t+1), r(t) \rangle_{t \in [\![0,T-1]\!]}$ is the projection into the MDP of the task episode. The goal is to generate trajectories with high discounted cumulative reward, also called more succinctly *return*: $\sum_{t=0}^{T-1} \gamma^t r(t)$. To do so, one needs to find a policy $\pi : \mathcal{X} \times \mathcal{A} \rightarrow [0, 1]$ which yields optimal expected returns. Formally, this means finding the policy which maximises the following function:

$$Q_\pi(x, a) \ = \mathbb{E}_\pi \left[\sum_{t' \geq t} \gamma^{t'-t} R(X_{t'}, A_{t'}) | X_t = x, A_t = a \right].$$

2.2 Problem setting

This section defines the Multi-Advisor RL (MAd-RL) framework for solving a single-agent RL problem. The n advisors are regarded as specialised, possibly weak, learners that are concerned with a sub part of the problem. Then an aggregator is responsible for merging the advisors' recommendations into a global policy. The overall architecture is illustrated in Figure 1. At each time step, an advisor j sends to the aggregator its local Q-values for all actions in the current state. The aggregator is defined with $f : \mathbb{R}^{n \times |\mathcal{A}|} \rightarrow \mathcal{A}$, which maps the received $\boldsymbol{q}_j$ values into an action of $\mathcal{A}$.

Inspired by Section 4.2 of Sun and Peterson [1999], we review ways to distribute a single-agent RL problem over several specialised advisors[1]:

1. State space approximation: each advisor has a local state space representation [Böhmer *et al.*, 2015; Laroche and Féraud, 2017]: $\mathcal{X}_j \subseteq \mathcal{X}$.

2. Segmentation of rewards: Separation of Concerns [van Seijen *et al.*, 2017] assumes that a complex task can be decomposed into subtasks defined by as many reward channels. Feudal RL [Dayan and Hinton, 1993; Vezhnevets *et al.*, 2017] learns to assign sub-goals under a recursive hierarchy master-slave. In both cases, advisor's reward functions R_j are local.

3. Criterion separation: sometimes, no objective function can clearly be designed, and the goal is to satisfy several criteria as much as possible. Multi-criteria RL [Gábor *et al.*, 1998] results in a segmentation of rewards, but with a specific aggregating policy.

4. Algorithm diversification [Wiering and Van Hasselt, 2008; Laroche and Féraud, 2017]: each algorithm intends to learn a global policy, but with different optimisations, hypotheses, regularizers and/or parameters.

5. Randomization: variance control through randomization of the learners can be obtained via instance sampling [Breiman, 1996], random initialisation [Glorot and Bengio, 2010], randomization of algorithms [Breiman, 2001], etc.

6. Sequencing of actions: each advisor is able to handle different sequences of actions. This is related to the options used in semi-MDP [Sutton *et al.*, 1999].

7. Factorisation of actions [Laroche *et al.*, 2009]: each advisor is responsible for a specific action dimension: for instance a robot might control its legs and its arms with different advisors.

[1] Items 6. and 7. are not developed further in this paper.

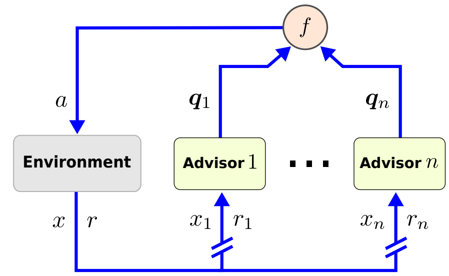


Figure 1: MAd-RL architecture

In summary, each advisor j is defined on a local representation $\phi_j : \mathcal{X} \rightarrow \mathcal{X}_j$, and its local state is denoted by $x_j = \phi_j(x) \in \mathcal{X}_j$. The advisor aims to find a an optimal evaluation of the state-action space in order to send the aggregator the most informative communication vector $\boldsymbol{q}_j = [Q_j(x_j, a)]_{a \in \mathcal{A}} = [Q_j(\phi_j(x), a)]_{a \in \mathcal{A}} \in \mathbb{R}^{|\mathcal{A}|}$; the state-action (x, a) values according to advisor j.

2.3 Separation of Concerns

MAd-RL falls within the Separation of Concerns (SoC) framework proposed in van Seijen *et al.* [2017]. Separation of Concerns distributes the responsibilities among several agents, that may communicate and have complex relationships, such as master-slave or collaborators-as-equal. This section transcribes under the MAd-RL notations the main theoretical result: a theorem ensuring, under conditions, that the advisors' training eventually converges.

Note that by assigning a stationary behaviour to each of the advisors, the sequence of random variables $X_0, X_1, X_2, \ldots$, with $X_t \in \mathcal{X}$ is a Markov chain. To formalize, let μ define a set of n stationary advisors and $\mathcal{M}$ be the space of all such sets. The following holds for all $\mu \in \mathcal{M}$:

$$\mathbb{P}(X_{t+1}|X_t, \mu) = \mathbb{P}(X_{t+1}|X_t, \dots, X_0, \mu).$$

For later analysis, we assume the following.

Assumption 1. *All the advisors environments are Markov:*

$$\mathbb{P}(X_{j,t+1}|X_{j,t}, A_t) = \mathbb{P}(X_{j,t+1}|X_{j,t}, A_t, \dots, X_{j,0}, A_0).$$

Theorem 1. *Under Assumption 1 and given any fixed aggregator, global convergence occurs if all advisors use off-policy algorithms that converge in the single-agent setting.*

Proof. Each advisor can be seen as an independent learner training from trajectories controlled by an arbitrary behavioural policy. If Assumption 1 holds, each advisor's environment is Markov and off-policy algorithms can be applied with convergence guarantees. □

Even though, Theorem 1 guarantees convergence, it does not guarantee the optimality of the converged solution. Moreover, this fixed point only depends on the SoC model and on the local bootstrapping methods (see Section 3), but not on the particular optimisation algorithms that are being used.

2.4 Aggregating advisors' recommendations

Finally, the f function's role is to aggregate the advisors' recommendations into a policy. These recommendations are expressed as their value functions $\boldsymbol{q}_j$. To our best knowledge, these recommendations enable to build any aggregator function encountered in the Ensemble methods literature [Dietterich, 2000a]: voting schemes [Gibbard, 1973], Boltzmann policy mixtures [Wiering and Van Hasselt, 2008] and of course value-function combinations [Sun and Peterson, 1999]. For the analysis, we restrict ourselves to the linear decomposition of the rewards:

$$R(x,a) \ = \sum_j w_j R_j(x_j, a),$$

which implies the same decomposition of return if they share the same γ. We define the global Q-function as follows:

$$Q(x,a) \ = \sum_j w_j Q_j(x_j, a).$$

The aggregator function is then defined in a greedy manner (during learning, exploration may be added):

$$f(x) = \operatorname*{argmax}_{a \in \mathcal{A}} Q(x,a).$$

This includes the setting where we have one advisor per reward channel [van Seijen *et al.*, 2017]: all w_j are equal to 1, and each advisor is specialised with a predefined local state space relevant to its task. This also models the setting of the weighting average of several learners on the global task: all w_j sum to 1 Sun and Peterson [1999].

2.5 MAd-RL as Ensemble Learning

MAd-RL can be interpreted as a framework for Ensemble Learning for RL. As such, a detailed positioning to previous work on this area has to be undertaken. Despite the widespread interest for Ensemble Learning [Dietterich, 2000a] and RL [Sutton and Barto, 1998], very little studies on Ensemble RL can be found. We provide their short survey hereinbelow.

Sun and Peterson [1999] use a boosting algorithm in a RL framework, but the boosting is performed upon policies, not RL algorithms. In this sense, this article can be seen as a precursor to the policy reuse algorithm [Fernández and Veloso, 2006] rather than Ensemble Learning.

Wiering and Van Hasselt [2008] combine five online RL algorithms on several simple RL problems and show that some mixture models of the five experts performs generally better than any single one alone. Their algorithms were off-policy, on-policy, actor-critics, etc. Faußer and Schwenker [2011] continue this effort in a very specific setting where actions are explicit and deterministic transitions. We show in Section 3 that the bootstrapping methods have meanings and that some recommendations can be made in accordance to the task definition.

In Harutyunyan *et al.* [2015], while all advisors are trained on different reward functions, these are potential based reward shaping variants of the same reward function. They are therefore embedding the same goals. As a consequence, it can be related to a bagging procedure. The advisors recommendation are then aggregated under the HORDE architecture [Sutton *et al.*, 2011], with local greedy off-policy bootstrapping. Two aggregator functions were tried out: majority voting and ranked voting. We call this method *local-max* and show in Subsection 3.1, that it induces critical theoretical shortcomings.

Finally, Laroche and Féraud [2017] follow a different approach in which, instead of boosting the weak advisors performances by aggregating their recommendation, they select the best advisor. This approach is beneficial for staggered learning, or when one or several advisors may not find good policies, but not for variance reduction brought by the committee, and it does not apply to compositional RL.

We believe that this article lays the theoretical foundation for Ensemble RL. Although the analysis provided in Section 3 is built on the linear composition of value functions, the same bootstrapping methods can be applied with the same distinctive features for any aggregator function, eg. majority/ranking voting, or Boltzmann policy aggregation.

3 Off-policy bootstrapping methods

This section present three different local off-policy bootstrapping methods. They differ in the policy they intend to evaluate and therefore intend to optimise different local Bellman equations. *local-max* evaluates the local greedy policy, *rand-policy* the random policy, and *agg-policy* the aggregator's greedy policy. They are presented and analysed under the linear composition aggregator presented in Subsection 2.4, but most considerations are also valid with other aggregating functions.

3.1 *Local-max* bootstrapping

An intuitive approach is to learn off-policy by bootstrapping on the locally greedy action: the advisor evaluates the local greedy policy. This bootstrapping method, referred to hereafter as *local-max*, has already been employed in Harutyunyan *et al.* [2015] and van Seijen *et al.* [2017]. Theorem 1 guarantees for each advisor j the convergence to the local optimal value function, denoted by Q_j^{lm}, which satisfies the Bellman optimality equation:

$$Q_j^{lm}(x_j, a) = \mathbb{E}\left[r_j\right] + \gamma \mathbb{E}\left[\max_{a' \in \mathcal{A}} Q_j^{lm}(x'_j, a')\right],$$

where the first term is the expectation of the local stochastic immediate reward function $r_j = R_j(x_j, a)$, and the second term is the future return expectation over the local stochastic transition function $P_j(x_j, a, x'_j)$. For the sake of simplicity, we removed the conditioning variables from the expectation notations hereafter. In the aggregator global view, we get:

$$\begin{aligned} Q^{lm}(x, a) &= \sum_j w_j Q_j^{lm}(x_j, a), \\ &= \sum_j \mathbb{E}\left[w_j r_j + \gamma w_j \max_{a' \in \mathcal{A}} Q_j^{lm}(x'_j, a')\right], \\ &= \mathbb{E}\left[\sum_j w_j r_j\right] + \gamma \mathbb{E}\left[\sum_j w_j \max_{a' \in \mathcal{A}} Q_j^{lm}(x'_j, a')\right]. \end{aligned}$$

By construction, $r = \sum_j w_j r_j$, and therefore we get:

$$\begin{aligned} Q^{lm}(x, a) &= \mathbb{E}\left[r\right] + \gamma \mathbb{E}\left[\sum_j w_j \max_{a' \in \mathcal{A}} Q_j^{lm}(x'_j, a')\right], \\ &\geq \mathbb{E}\left[r\right] + \gamma \mathbb{E}\left[\max_{a' \in \mathcal{A}} Q^{lm}(x', a')\right]. \end{aligned}$$

Local-max suffers from an inversion between the $\max$ and $\sum$ operators and it tends as a consequence to overestimate the state-action values, in particular when the advisors disagree on the optimal action. This flaw has critical consequences in practice. In particular, it creates a lot of *attractor* situations. We'll see in Section 4.2 that these situations are

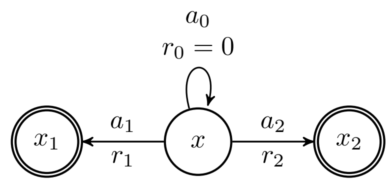


Figure 2: Attractor example.

encountered in the Pac-Boy experiment, and we believe in most other RL tasks. Before coming to a general definition, let us explain the attractor phenomenon on the MDP example in Figure 2. In the central state, the system has three possible actions: stay put (action a_0), perform advisor 1's goal (action a_1), or perform advisor 2's goal (action a_2). Once achieving a goal, the trajectory ends. The Q-function values for each action are easy to compute:

$$\begin{aligned} Q^{lm}(s, a_0) &= \mathbb{E}\left[r\right] + \gamma \mathbb{E}\left[\sum_j \max_{a' \in \mathcal{A}} Q_j^{lm}(x'_j, a')\right], \\ &= 0 + \gamma r_1 + \gamma r_2, \\ &= \gamma r_1 + \gamma r_2, \\ Q^{lm}(s, a_1) &= r_1, \\ Q^{lm}(s, a_2) &= r_2. \end{aligned}$$

As a consequence, if $\gamma > r_1/(r_1+r_2)$ and $\gamma > r_2/(r_1+r_2)$, the aggregator's policy after convergence will be to execute action a_0 *sine die*. This may have some apparent similarity with the Buridan's ass paradox [Rescher, 2005]: a donkey is equally thirsty and hungry and cannot decide to eat or to drink and dies of its inability to make a decision because of the determinism of judgement stated in antic philosophy. Nevertheless, the *local-max* sub-optimality does not come from actions that are equally good, nor from the determinism of the policy, since adding randomness to the system will not help (see Section 1.6 in Zbilut [2004]). Now, let us define more generally the concept of attractors.

Definition 1. *An attractor x is a state where the following inequality holds:*

$$\max_{a \in \mathcal{A}} \sum_j w_j Q_j^{lm}(x_j, a) \leq \gamma \sum_j w_j \max_{a \in \mathcal{A}} Q^{lm}(x_j, a).$$

An attractor is a state where *local-max* would lead the aggregator to stay in that state, if it had the chance to. Note that there is no condition on the existence of actions allowing the system to be actually static. Indeed, the system might be stuck in an attractor set, keep moving, but opt to never achieve its goals. To understand how this may happen, just replace the middle state x in Figure 2 with an indefinitely large set of similar attractors: where action a_0 performs a random transition in the attractor set, and actions a_1 and a_2

respectively achieve tasks of advisors 1 and 2. Moreover, it may happen that an attractor set is escapable by the lack of actions keeping the system in an attractor set. For instance, in Figure 2, if action a_0 is not available, the central state remains an attractor, but an unstable one.

Definition 2. *An advisor j is said to be monotonic if the following condition is satisfied:*

$$\forall x_j \in \mathcal{X}_j, \forall a \in \mathcal{A}, \quad Q_j^{lm}(x_j, a) \geq \gamma \max_{a' \in \mathcal{A}} Q_j^{lm}(x_j, a').$$

The intuition behind the monotonic property is that no action is worse than losing one turn to do nothing. In other words, only progress can be made towards this task, and therefore *noop* actions are regarded by this advisor as the worst ones.

Theorem 2. *If all the advisors are monotonic, there cannot be any attractor.*

Proof. Let sum Definition 2 over advisors:

$$\sum_j w_j Q_j^{lm}(x_j, a) \geq \gamma \sum_j w_j \max_{a' \in \mathcal{A}} Q_j^{lm}(x_j, a'),$$
$$\max_{a' \in \mathcal{A}} \sum_j w_j Q_j^{lm}(x_j, a') \geq \sum_j w_j Q_j^{lm}(x_j, a),$$

which proves the theorem. □

The condition stated in Theorem 2 is very restrictive. Most of RL problems do not fall into this category, even for small γ values. Navigation tasks do not qualify by nature: when the system goes into a direction that is opposite to some goal, it gets into a state that is worse than staying in the same position. As well, Theorem 2 does not apply to RL problems with states that terminate the trajectory while some goals are still incomplete. Still, there exist some RL problems where Theorem 2 can be applied, such as resource scheduling where each advisor is responsible for the progression of a given task. Note that a MAd-RL setting without any attractors does not guarantee optimality for *local-max*. An attractor-free setting simply means that the system will continue making progress towards goals as long as there are any opportunity to do so.

3.2 *Rand-policy* bootstrapping

For any MAd-RL problem, there exists other off-policy bootstrapping methods guaranteed to be attractor free. The advisors need not bootstrap their value function on the basis of local policies, but rather on a shared reference. A potential reference policy may be the random policy over the action set $\mathcal{A}$. This Q-function bootstrapping method is called *rand-policy* hereafter. Once again, Theorem 1 guarantees the convergence of the local optimisation process to its local optimal value, denoted by Q_j^{rp}, which satisfies the following Bellman equation:

$$Q_j^{rp}(x_j, a) = \mathbb{E}\left[r_j + \frac{\gamma}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} Q_j^{rp}(x_j', a') \right],$$
$$Q^{rp}(x, a) = \mathbb{E}\left[r + \frac{\gamma}{|\mathcal{A}|} \sum_j w_j \sum_{a' \in \mathcal{A}} Q_j^{rp}(x_j', a') \right],$$
$$= \mathbb{E}\left[r + \frac{\gamma}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} Q^{rp}(x', a') \right].$$

The local *rand-policy* optimisation is equivalent to the global *rand-policy* optimisation. As such, it does not suffer from local attractor issue described in Section 3.1. However, optimising the value function with respect to the random policy is in general far from the optimal solution to the global MDP problem.

3.3 *Agg-policy* bootstrapping

Another solution is to use the aggregator's policy as the reference. In this method, referred to as *agg-policy*, the aggregator is in control, and the advisors are evaluating the current aggregator's greedy policy f with respect to their local focus. Theorem 1 does not apply here because the aggregator's policy is dependent on the other advisors, which means that, even though the environment can still be modelled as an MDP, the training procedure is not. Assuming that all advisors jointly converge to their respective local optimal value, denoted by Q_j^{ap}, it satisfies the following Bellman equation:

$$Q_j^{ap}(x_j, a) = \mathbb{E}\left[r_j + \gamma Q_j^{ap}(x_j', f(x')) \right],$$
$$Q^{ap}(x, a) = \mathbb{E}\left[r + \gamma \sum_j w_j Q_j^{ap}(x_j', f(x')) \right],$$
$$= \mathbb{E}\left[r + \gamma Q^{ap}(x', f(x')) \right],$$
$$= \mathbb{E}\left[r + \gamma Q^{ap}(x', \operatorname*{argmax}_{a' \in \mathcal{A}} Q^{ap}(x', a')) \right],$$
$$= \mathbb{E}\left[r + \gamma \max_{a' \in \mathcal{A}} Q^{ap}(x', a') \right].$$

It is interesting to notice that this global Bellman equation is actually the global Bellman optimality equation. We can therefore conclude its uniqueness but unfortunately, as aforementioned, this comes with the non-Markovian property of the aggregator's policy f at the local learner scope. As a result, local learners are not guaranteed to converge in the general case. Nevertheless, it can be proven in a limit-case:

Theorem 3. *If, for all advisors j, $\mathcal{X}_j = \mathcal{X}$, using SARSA [Rummery and Niranjan, 1994] update rule for each advisor with respect to the aggregator's maximising action* *is equivalent to applying Q-learning [Watkins, 1989] update rule on the global agent.*

Proof. Let $\bar{a}_{x'}$ denote the aggegator's greedy policy action in state x'. The Q-learning update rule for the global agent can be decomposed as follows:

$$\begin{aligned}Q^{ap}(x,a) &\leftarrow (1-\alpha)Q^{ap}(x,a) + \alpha\left[r + \gamma Q^{ap}(x', \bar{a}_{x'})\right],\\ &= (1-\alpha)\sum_j w_j Q_j^{ap}(x,a)\\ &\quad + \alpha\left[\sum_j w_j r_j + \gamma \sum_j w_j Q_j^{ap}(x', \bar{a}_{x'})\right].\end{aligned}$$

The SARSA update rule for each advisor with respect to the aggregator's action $\bar{a}_{x'}$ is written as follows:

$$\begin{aligned}Q_j^{ap}(x_j,a) \leftarrow (1-\alpha)Q_j^{ap}(x_j,a) + \alpha\big[r_j\\ + \gamma Q_j^{ap}(x'_j, \bar{a}_{x'})\big].\end{aligned}$$

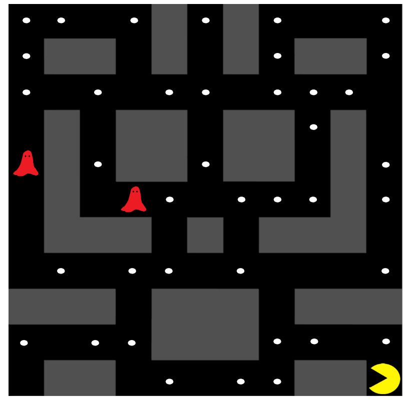
Since $x_j = x$ and $x'_j = x'$ from the theorem's assumptions, we notice that the update rule of the global Q-learning algorithm is the sum of the update rule over the local SARSA advisors with respect to the aggregator's action, which proves the theorem. □

4 Experiment and results

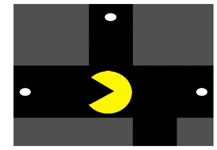
4.1 Pac-Boy

We evaluate a MAd-RL model on a simplified version of Ms. Pac-Man, which we call Pac-Boy (see Figure 3a). Ms. Pac-Man is considered as one of the hardest games from the Atari benchmark set [Mnih *et al.*, 2015].

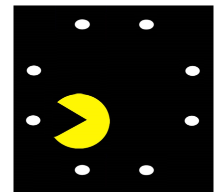
Pac-Boy navigates in a 11x11 maze with a total of 76 possible positions and 4 possible actions in each state: $\mathcal{A} = \{N, W, S, E\}$, respectively for North, West, South and East. Bumping into a wall simply causes the player not to move without penalty. Since Pac-Boy always starts in the same position, there are 75 potential fruit positions. The fruit distribution is randomised: at the start of each new episode, there is a 50% probability for each position to have a fruit. A game lasts until the last fruit has been eaten, or after the 300$^{\text{th}}$ time step. During an episode, fruits remain fixed until they get eaten by Pac-Boy. As in Ms. Pac-Man, ghosts are preventing Pac-Boy from eating all the fruits. However, due to the smaller grid, there are only two of them and they both move randomly. The state of the game consists of the positions of Pac-Boy, fruits, and ghosts: $76 \times 2^{75} \times 76^2 \approx 10^{28}$ states. Hence, no global representation system can be implemented without using function approximation. Pac-Boy gets a $+1$ reward for every eaten fruit and a -10 penalty when it is touched by a ghost.



(a) Pac-Boy full map



(b) 3 fruits & a wall



(c) No *noop* action

Figure 3: The Pac-Boy game. On the left, the full map: Pac-Boy is yellow, the corridors are in black, the walls in grey, the fruits are the white dots amd the ghosts are in red. On the right, two examples of attractors in the Pac-Boy domain.

Experiments – Time scale is divided into 50 epochs lasting 20,000 transitions each. At the end of each epoch an evaluation phase is launched for 80 games. Each experimental result is presented along two dimensional performance indicators: the averaged non discounted rewards and the average length of the games. The average non discounted rewards can be seen as the number of points obtained in a game. Its theoretical expected maximum is 37.5 and the random policy average performance is around -80 (being eaten about 10 times by the ghosts).

MAd-RL Setup – Each advisor is responsible for a specific source of reward (or penalty). More precisely, we separate the concerns as follows: we assign an advisor to each possible fruit location. This advisor sees a +1 reward only if a fruit at its assigned position gets eaten. Its state space consists of Pac-Boy's position, resulting in 76 states. In addition, we assign an advisor to each ghost. This advisor receives a -10 reward if Pac-Boy bumps into its assigned ghost. Its state space consists of Pac-Boy's position and the ghost's position, resulting in 76^2 states. A fruit advisor is only active when there is a fruit at its assigned position. Because there are on average 37.5 fruits, the average number of advisors running at the beginning of each episode is 39.5. Each fruit advisor is set inactive when its fruit is eaten.

The learning was performed through Temporal Difference updates. Due to the small state spaces for the advisors, we can use a tabular representation. We train all learners in parallel with off-policy learning, with Bellman residuals computed as presented in Section 3 and a constant $\alpha = 0.1$ parameter. The aggregator function sums the Q-values for each action $a \in \mathcal{A}$: $Q(x,a) := \sum_j Q_j(x_j, a)$, and uses ϵ-greedy action selection with respect to these summed values. Because ghost agents have exactly identical MDP, we also benefit from direct knowledge by sharing their Q-tables.

One can notice that Assumption 1 holds in this setting and that, as a consequence, Theorem 1 applies for *local-max* and *rand-policy* .

Baselines – Our first baseline is a system that uses the exact same input features as the MAd-RL model. Specifically, the state of each advisor of the MAd-RL model is encoded with a one-hot vector and all these vectors are concatenated, resulting in a sparse binary feature vector of size $17,252$ with about 40 active features per time step. This vector is used for linear function approximation with Q-learning. We refer to this setting with *linear Q-learning*.

We then consider two deep RL baselines. The first is the standard DQN algorithm [Mnih *et al.*, 2015] with reward clipping (referred to as *DQN-clipped*). The second is Pop Art [van Hasselt *et al.*, 2016], which can be combined with DQN in order to handle large magnitudes of reward (referred to as *DQN-scaled*). The input to both *DQN-clipped* and *DQN-scaled* is a 4-channel binary image, where each channel is in the shape of the game grid and represents the positions of one of the following features: the walls, the ghosts, the fruits, or Pac-Boy.

4.2 Attractors in navigation domains

Regarding the weakness of *local-max*, an analysis of situations where attractors occur in the Pac-Boy domain must be performed. The three-fruit attractor illustrated in Figure 3b happens when the system is in a state with equal distance between three fruits and adjacent to a wall, enabling it to perform a *noop* action. Moving towards a fruit, makes it closer to one of the fruits, but further from the two other fruits, since diagonal moves are not allowed. Expressing the real value of each action under *local-max* gives the following:

$$Q^{lm}(x,S) = \gamma \sum_j \max_{a\in\mathcal{A}} Q_j^{lm}(x_j,a) = 3\gamma^2,$$

$$Q^{lm}(x,N) = Q^{lm}(x,E) = Q^{lm}(x,W) = \gamma + 2\gamma^3.$$

That means that, if $\gamma > 0.5$, $Q^{lm}(x,S) > Q^{lm}(x,N) = Q^{lm}(x,E) = Q^{lm}(x,W)$. As a result, the aggregator would opt to go South and hit the wall indefinitely. This is a practical example of an attractor as defined in Subsection 3.1. In the case of Figure 3b, there is a *noop* action, and given the number of corridors, it is difficult to find a Pac-Boy situation without a wall enabling a static action. Nevertheless, as Section 3.1 predicts, the attractors can be encountered in navigation tasks even in settings without any *noop* action as in Figure 3c, where the player is placed in a 2x2 square with 8 fruits surrounding it. The action-state values of the aggregator under *local-max* are:

$$Q^{lm}(x,N) = Q^{lm}(x,E) = 2\gamma + 4\gamma^2 + 2\gamma^3,$$

$$Q^{lm}(x,S) = Q^{lm}(x,W) = 1 + \gamma + \gamma^2 + 3\gamma^3 + 2\gamma^4.$$

Once again, that means that, if $\gamma > 0.5$, $Q^{lm}(x,N) = Q^{lm}(x,E) > Qv(x,S) = Q^{lm}(x,W)$. After moving North or East, the system arrives in a state that is symmetrically equivalent to the first one. More generally in a deterministic[2] navigation task like Pac-Boy where each action a in a state x can be cancelled by a new action a_x^{-1}, it can be shown that the condition on γ is a function of the size of the action set $\mathcal{A}$.

Theorem 4. *State $x \in \mathcal{X}$ is guaranteed not to be an attractor if all these conditions are satisfied:*

- $\forall a \in \mathcal{A}, \exists a_x^{-1} \in \mathcal{A}$, *such that* $P(P(x,a), a_x^{-1}) = x$,
- $\forall a \in \mathcal{A}, R(x,a) \geq 0$,
- *and* $\gamma \leq \dfrac{1}{|\mathcal{A}|-1}$.

Proof. Let us denote $\mathcal{J}_a^x$ as the set of advisors for which action a is optimal in state x. $Q_a^{lm}(x)$ is defined as the sum of perceived value of performing a in state x by the advisors that would choose it:

$$Q_a^{lm}(x) = \sum_{j\in\mathcal{J}_a^x} w_j Q_j^{lm}(x'_j, a).$$

Let a^+ be the action that maximises this $Q_a^{lm}(x)$ function:

$$a^+ = \operatorname*{argmax}_{a\in\mathcal{A}} Q_a^{lm}(x).$$

We now consider the left hand side of the inequality characterising the attractors in Definition 1:

$$\max_{a\,\in\mathcal{A}} \sum_j w_j Q_j^{lm}(x_j,a) \geq \sum_j w_j Q_j^{lm}(x_j,a^+),$$

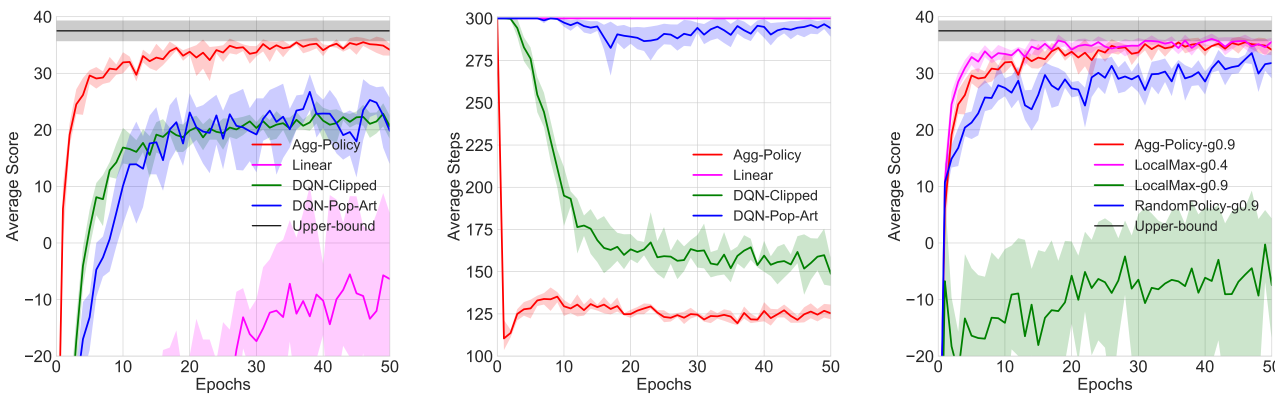
$$= Q_{a^+}^{lm}(x) + \sum_{j\notin\mathcal{J}_{a^+}^x} w_j Q_j^{lm}(x_j,a^+),$$

$$= Q_{a^+}^{lm}(x) + \sum_{j\notin\mathcal{J}_{a^+}^x} w_j \left(R(x,a^+) + \gamma \max_{a'\in\mathcal{A}} Q_j^{lm}(x'_j,a') \right).$$

Since $R(x,a^+) \geq 0$, and since the a' maximising $Q_j^{lm}(x'_j,a')$ is at least as good as the cancelling action $(a^+)_x^{-1}$, we can follow with:

$$\max_{a\,\in\mathcal{A}} \sum_j w_j Q_j^{lm}(x_j,a) \geq Q_{a^+}^{lm}(x) + \sum_{j\notin\mathcal{J}_{a^+}^x} w_j \gamma^2 \max_{a\in\mathcal{A}} Q_j^{lm}(x_j,a).$$

[2] A more general result on stochastic navigation tasks can be demonstrated. We limited the proof to the deterministic case for the sake of simplicity.



(a) Avg scores against baselines (b) Avg episode length against baselines (c) Avg scores for different methods

Figure 4: Empirical results in the regular setting comparing MAd-RL with baselines, and different bootstrapping methods.

By comparing this last result with the right hand side of Definition 1, the condition for x not being an attractor becomes:

$$(1-\gamma)Q^{lm}_{a^+}(x) \geq (1-\gamma)\gamma \sum_{j \notin \mathcal{J}^x_{a^+}} w_j \max_{a \in \mathcal{A}} Q^{lm}_j(x_j, a),$$

$$Q^{lm}_{a^+}(x) \geq \gamma \sum_{a \neq a^+} \sum_{j \in \mathcal{J}^x_a} w_j Q^{lm}_j(x_j, a),$$

$$Q^{lm}_{a^+}(x) \geq \gamma \sum_{a \neq a^+} Q^{lm}_a(x).$$

It follows directly from the inequality $Q^{lm}_{a^+}(x) \geq Q^{lm}_a(x)$, that $\gamma \leq 1/(|\mathcal{A}|-1)$ guarantees the absence of attractor. □

Theorem 4 determines sufficient conditions for not having any attractor in the MDP. In the Pac-Boy domain, the cancelling action condition is satisfied for every $x \in \mathcal{X}$. As for the γ condition, it is not only sufficient but also necessary, since being surrounded by goals of equal value is an attractor if $\gamma > 1/3$. In practice, an attractor becomes stable only when there is an action enabling it to remain in the attraction set. Thus, the condition for not being stuck in an attractor set can be relaxed to $\gamma \leq 1/(|\mathcal{A}|-2)$. Hence, the result of $\gamma > 1/2$ in examples illustrated by Figures 3b and 3c. It is still a very restrictive condition, considering that most navigation problems have at least the four cardinal-point actions.

Notice that there exists many navigation problems where the assumption of cancelling actions does not hold. For instance a car on the top of a hill with two equal goals on each side of the hill would go faster moving down the hill than up. As a consequence, even if the car has only three actions: $\{left, noop, right\}$, the *local-max* aggregator would be stuck in the attractor, the hill, by repeating the *noop* action.

4.3 Results

The results of this experiment are presented in Figures 4a, 4b and 4c. Seven different settings have been compared: the three baselines *linear Q-learning*, *DQN-clipped*, and *DQN-scaled*, and four MAd-RL settings: *local-max* with $\gamma = 0.4$, *local-max* with $\gamma = 0.9$, *rand-policy* with $\gamma = 0.9$, and *agg-policy* with $\gamma = 0.9$.

Figure 4a shows that *linear Q-learning* performs the worst. It benefits from no state space reduction and cannot generalize as well as the Deep RL baselines: *DQN-clipped* and *DQN-scaled*, which perform better but do not progress after reaching a reward close to 20. Despite their similar results with respect to performance, Figure 4b reveals that their learnt policies are in fact very different. *DQN-scaled* is much wearier of the high negative reward obtained from being eaten by the ghosts and thus takes much more time to eat all the fruits.

MAd-RL settings perform considerably better. The comparison between the bootstrapping methods on Figure 4c reveals that, with small γ value, *local-max* is very efficient: the best in our benchmark. However, *local-max* performs awfully with $\gamma = 0.9$. This is empirical confirmation that the theoretical drawbacks obtained in Sections 3.1 and 4.2 are also a practical issue. Attractors aside, the small γ value does not have a big impact on the game performance in the Pac-Boy domain for both fruit collection and ghost avoidance. The fruit collection problem is similar to the travelling salesman problem, which is known to be NP-complete [Papadimitriou, 1977]. But, the suboptimal small-γ policy consisting of moving towards the closest fruits is in fact a decent one. Regarding the ghost avoidance, *local-max* with small γ gets an advantage over other settings: the local optimisation guarantees a perfect control of the system near the ghosts. On the contrary, in the two other settings, *rand-policy* and *agg-policy*, the ghost advisor is uncertain of the next action of their bootstrapping policies. As a result, they become more conservative around the ghosts, especially *rand-policy* that considers each future action as equally likely.

Regarding *agg-policy*, even though its performance remains near that of *local-max*, it still suffers from the fact that the local learners cannot fully make sense of the aggregator's actions due to their limited state space representations, which transgresses Theorem 3's assumptions. We tested other γ values for *agg-policy* and a value close to 0.4 was slightly better. This is a good trade-off between the long-term horizon and the noise in the Q-function propagated by high values of γ. More precisely, a smaller γ makes the ghost advisors less fearful of the ghosts, which is profitable when collecting the nearby fruits.

4.4 Results with noisy rewards

But, even if using a very small γ does the trick, especially for *local-max* in the Pac-Boy game, it can be dangerous in other environments. The reason is that the objective function gets distorted and even more importantly the reward signal diminishes exponentially as a function of the distance to the goal, which might have critical consequences in a noisy environment, hence this subsection experiment.

Several levels of Gaussian white noise η_σ with standard deviation $\sigma \in \{0.01, 0.1\}$ have been applied to the reward signal: at each turn, each advisor now receives $\hat{r}_j = r_j + \eta_\sigma$ instead. Since the noise is white, the Q-functions remain the same, but their estimators obtained during sampling is corrupted by noise variance. We expect that small γ values cause the reward signal to be overwhelmed by this noise, and high γ values, while they propagate the noise further, also propagate the reward signal in such a way that they should be more robust. Indeed, the amplitude of non-zero rewards are much bigger than those of the noise terms.

Empirical results displayed in Figure 5 confirm this: *agg-policy* performs better than *local-max* even under noise with variance σ^2 100 times larger. Indeed, the fruit advisors are only able to perceive the fruits that are in a radius dependent on γ and σ, a smaller γ implying a smaller radius. *Local-max*, incompatible with high γ values, is therefore myopic and cannot perceive distant fruits. The same kind of limitations are expected to be encountered for small γ values when the local advisors rely on state approximations, and/or when the transitions are stochastic. Also, we recall here that optimising with respect to an artificial γ value might converge to policies that are largely suboptimal regarding the true γ value in the objective function.

It is worth mentioning here that hybrid settings with *local-max* for the ghost advisors and *agg-policy* for the fruit advisors achieve very good performance, even with high γ. This is due to the fact that the stale positions caused by attractors cannot be encountered with only ghost advisors, which apply a repulsion mechanism and not a goal in of itself.

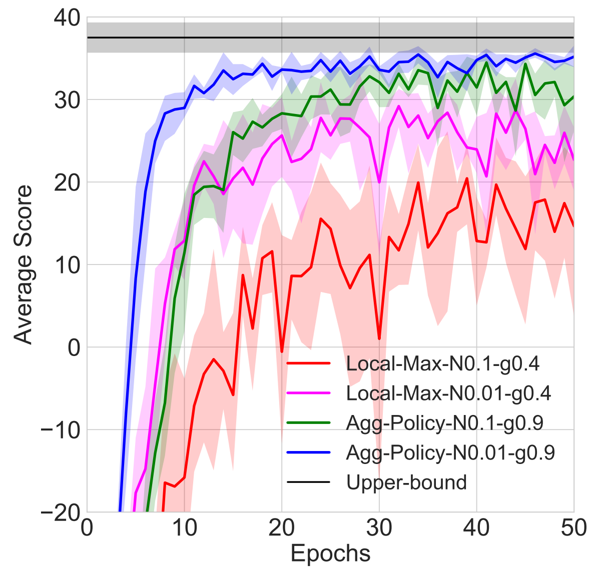


Figure 5: Average performance with noisy rewards

5 Conclusion and future work

This article presented MAd-RL, a novel framework for decomposing a single-agent RL problem into simpler problems tackled by learners called advisors. Then, we showed that the advisors can be trained according to different bootstrappings: *local-max* bootstraps with the local greedy action. It is guaranteed to converge but we demonstrate that a $\sum \max$ inversion causes its optimal policy to be endangered by attractors. *Rand-policy* bootstraps with respect to the random policy. It is also guaranteed to converge and is robust to attractors, but its random bootstrapping prevents the advisors from planning in an efficient way. Finally, *agg-policy* bootstraps with respect to the aggregator's policy. It optimises the system according to the global Bellman optimality equation, but without any guarantee of convergence.

All bootstrapping methods are compared on the Pac-Boy domain, where the reward function can be decomposed in an efficient way. It is shown that the attractors prevent us from using *local-max* with high γ values, but that *local-max* with small γ values still performs best. *Agg-policy* is almost as good and can be employed with high γ values. We then show that adding noise to the environment disrupts the training with small γ values, and as a result, *agg-policy* performs the best.

As future work, we plan on working on learning the aggregating function in domains that are less straightforwardly compositional. Also, similarly to McGovern and Barto [2001] or Vezhnevets *et al.* [2017] for options, we are interested in discovering the reward function decomposition, and to then distribute the rewards per independent advisors, which would infer their own local state space $\mathcal{X}_i$.

References

Wendelin Böhmer, Jost T Springenberg, Joschka Boedecker, Martin Riedmiller, and Klaus Obermayer. Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *KI-Künstliche Intelligenz*, 2015.

Leo Breiman. Bagging predictors. *Machine learning*, 1996.

Leo Breiman. Random forests. *Machine learning*, 2001.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Proceedings of the 7th Annual Conference on Neural Information Processing Systems (NIPS)*, 1993.

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, 2000.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 2000.

Stefan Faußer and Friedhelm Schwenker. Ensemble methods for reinforcement learning with function approximation. In *International Workshop on Multiple Classifier Systems*. Springer, 2011.

Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Conference on Autonomous Agents and Multi-Agent Aystems (AAMAS)*, 2006.

Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *Proceedings of the 15th International Conference on Machine Learning (ICML)*, 1998.

Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica: journal of the Econometric Society*, 1973.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Anna Harutyunyan, Tim Brys, Peter Vrancx, and Ann Nowé. Off-policy reward shaping with ensembles. *arXiv preprint arXiv:1502.03248*, 2015.

Romain Laroche and Raphaël Féraud. Algorithm selection of off-policy reinforcement learning algorithm. *arXiv preprint arXiv:1701.08810*, 2017.

Romain Laroche, Ghislain Putois, Philippe Bretier, and Bernadette Bouchon-Meunier. Hybridisation of expertise and reinforcement learning in dialogue systems. In *Proceedings of the 9th Annual Conference of the International Speech Communication Association (Interspeech)*, 2009.

Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

Christos H Papadimitriou. The euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 1977.

Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Proceedings of the 11th Advances in Neural Information Processing Systems (NIPS)*, 1998.

Nicholas Rescher. *Cosmos and Logos: Studies in Greek Philosophy*. Topics in Ancient Philosophy / Themen der antiken Philosophie. De Gruyter, 2005.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. Technical report, Technical report, Stanford University, 2003.

Ron Sun and Todd Peterson. Multi-agent reinforcement learning: weighting and partitioning. *Neural networks*, 1999.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceedings of the 10th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Proceedings of the 29th Advances in Neural Information Processing Systems (NIPS)*, 2016.

Harm van Seijen, Mehdi Fatemi, Joshua Romoff, and Romain Laroche. Separation of concerns in reinforcement learning. *CoRR*, abs/1612.05159v2, 2017.

Alexander Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Christopher JCH Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 2008.

Joseph P Zbilut. *Unstable singularities and randomness: Their importance in the complexity of physical, biological and social sciences*. Elsevier, 2004.