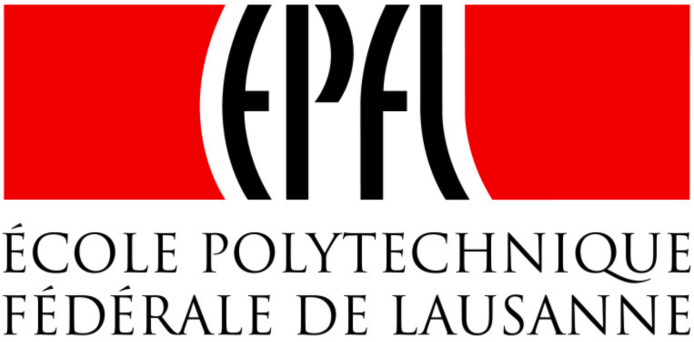
ÉCOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
MT SECTION
FAURY Louis

SEMESTER PROJECT REPORT

LASA



Learning from noisy demonstrations: the role of compliance in exploration-exploitation trade-off

Laboratory	Learning Algorithms and Systems Laboratory
Supervisers	Mahdi Khoramshahi, Sutcliffe Andrew
Semester	Spring

Contents

Chapter 1

Introduction

1.1 Motivations

When it comes to real world manipulation tasks, common machine learning algorithms are sometimes unable to come up with feasible solutions, or at least fail to do so in an acceptable computational time. A natural way to accelerate the learning process is to provide a learning algorithm with prior belief on its environment, as well as recommandations with respect to the task it is trying to learn.

Such prior belief is often achieved thanks to demonstrations, performed by a human teacher. If this approach is largely covered in the literature (see 1.2.1), it is not really clear how to learn from a suboptimal teacher, regardless of its level of sub-optimality. Such learning abilities could enlarge the human / robot interactions possibilities, as only little knowledge in robotics or artificial intelligence would be needed to help a robot learn a task.

1.1.1 Motivating examples

For instance, let's consider the example of a robotic arm learning to grasp an object : an unexperienced teacher might provide it with demonstrations that operate near the robot's workspace edges (which is often undesirable in robotics). Even if the robotic arm should not trust this demonstration if it wants to learn an optimal solution, simply discarding it would be unwise, since it contains many important informations relative to the task (pose of the object, joint coordination,..). We must therefore find a way to handle the demonstration in order to infer those parameters without falling in its suboptimal reproduction.

Before going any further, let us rephrase the previous problem slightly differently. Indeed, we can draw a parallel between the latest example and the one of a children learning to dance (or any other technical skill). Because the child's dance teacher doesn't have the same physical abilities than the child, he might give the child a suboptimal way (with respect to the child's physical abilities) of performing dance moves. In a ideal learning process, the child would use the prior information given by the teacher, and use it to practice its skill. He will soon be aware of its own abilities, and can then start learning by itself, exploring around what he has learnt so far. He might find that, by slightly changing how he performs some moves, he is able to dance better than by blindly listening to the teacher. Therefore, the learning child will first be *compliant* with the teacher, before trying things out by itself once it has become pretty good at performing the learned skill.

1.1.2 Goal

This semester project aims at introducing a theoretical formulation of such a compliance-based behavior, and experimentally test its performance on simple problems.

The underlying goal behind this objective is to get better intuition about how interactive learning between humans and robots can be achieved. Indeed, we would like to be able to teach a robot from demonstration not by only by providing it a large number of trajectories that he will then use as a motion generator, but by directly interacting with him. By showing a robotic arm, at different times of its learning procedure, concrete examples on how to avoid its closest obstacle and grab a given object, seems like a more natural way of teaching the robot a task, while still enabling him to correct some small sub-optimality we could have introduced to it.

This approach also allows us to tackle a somehow different problem, related to transfer learning. Indeed, the teacher might not be a human but another learner, only better trained than the current learner. In such a case, we would like the learner to quickly find if it can trust its mentor, and if not where it should focus its computations to overcome its mentor's sub-optimality. Such questions will therefore be tackled in this report.

1.2 Background

We hereinafter introduce some notions from the Learning from Demonstration (LfD) framework, as well as some transfer learning metrics that will show useful later on.

1.2.1 Learning from demonstration

Learning from demonstration (LfD) is a framework where a robot can learn from interacting with a human. It particularly focus on the case where a mentor (human) provides demonstration on how to perform a task. Such an approach is also known as Programming from Demonstration (PbD) or Imitation Learning.

LfD mostly lies on the principle that the learning robot can be taught new tasks by end-users, without having to be programmed again. We therefore are dealing with robots that are able to generalize from demonstration, be it only to understand what is the task that they must learn. They are many underlying problematics and opened questions on the subject, detailed in [BCD16].

Many LfD approaches are combined with reinforcement learning techniques. This is mostly related to the understanding that demonstrations can easily speed-up the reinforcement learning process by providing a finite or infinite set of good solutions. However, this assume some kind of reproducibility by the robot, as well as a similar enough context between demonstration and reproduction.
Under such assumptions, reinforcement learning techniques can use LfD to speed up their process and discover new control policies through a directed exploration of its action-state space. Indeed, demonstrations are used to guide the reinforcement learning algorithm's exploration, reducing the time and iterations needed for an agent to discover a new control policy, and eventually overcoming the teacher's policy sub-optimalities.

They are many ways of performing such a demonstration-based learning. We remind here a few of them, taken from [ACVB09], all related to Imitation Reinforcement Learning (IRL) :

- policy derivations techniques : directly approximate the robot's mapping from state to action by reproducing and generalizing the teacher's one.
- use demonstration data to learn a model from the world's dynamic (or transition probability model) to compute the optimal policy (see [PB03]).
- use demonstration data and additional user intention to learn the rule that associate a set of pre and post-conditions which each action as well as a sparse transition model to plan for actions.

The framework we described hereinbefore can easily be extended to more than one teacher. Moreover, such teachers can actually be other learning algorithms with different levels of training. Quickly, the problem drifts toward the underlying issue of performing transfer skill between agents. As this will also be tackled during this project, the next section reminds a few useful facts about the theory of transfer learning.

1.2.2 Transfer learning

Transfer learning deals with speeding a learning process thanks to another learning experience - for instance training an agent based on the previous training on another agent, performing a similar task. The literature of transfer learning in reinforcement learning is pretty large, and for a full description of the different problematics (homogeneous / inhomogeneous sets of actions, multi task learning, ..) we'll refer to [TS09].

As we said before, one of our goal is to assess our method as a transfer learning one, and up to a point, consider our teacher to be a trained (fully or not) agent, before considering many different agents as potential mentors. We would consider a really simple case of transfer learning, where all agents display the same state-action state, with the same abilities and constraints.

Hence, we remind here some metrics to evaluate transfer learning methods that we would have to use to compare our different approaches :

- *Jump start* : the initial performance of an agent can be improved by transfer.
- *Asymptotic performance* : how close is the asymptotic learned policy from the optimal policy ?
- *Total or averaged reward* : averaged reward may be improved by transfer
- *Transfer ratio* : Ratio of total reward accumulated by the transfer learner and the total of reward accumulated by a non-transfer learner.
- *Time to threshold* : the time required by a transfer learning to reach a pre-specified level of skills in achieving its task.

All those metrics actually compared different aspects of how well a RL algorithm learns. Comparing two reinforcement learning methods is actually still an open question, and hence we'll need to address several of those metrics to decide on the wellness of our derived TL methods.

1.3 Outline

To grasp ideas and intuitions about a compliant-based imitation learning method, we are going to start with a fairly simple environment and an explicit task. We hope that a simple enough state space will allow us to better understand how compliance in learning by demonstration should be tackled with a reinforcement learning formulation. We also hope we would be able to generalize to more complex situations once the understanding on a simple but generic model is mastered.

Hence, after defining a simple two dimensional grid-world state space with a simple action set, we will quickly study how well the different classical reinforcement learning policy search algorithms performs on such a space. We will then introduce new exploration policies, based on the confidence the learner has in its teacher and compare their efficiency with the latter algorithms. We will also focus on the relation between the learner and the prior information that its mentor's recommandations represent. Especially, we will study how well a learner can overcome its mentor own sub-optimality, focusing on largely suboptimal mentors. We call *largely* suboptimal a teacher providing demonstrations with potential danger or obvious downsides for the learner.

Chapter 2

Reinforcement Learning

The reinforcement learning approach being an essential aspect of this project, this chapter is intended to remind the foundations of the reinforcement learning theory and its practical implementations.

2.1 Formulation

2.1.1 Definitions

Reinforcement learning is a framework in which an *agent* (or a *learner*) learns its actions from interaction with its environment. The environment generates scalar values called *rewards*, that the agent is seeking to maximize over time.

Let $\mathcal{S}$ denote the state space in which our agent evolves (the localization of a robot on a grid for instance), and $\forall s \in \mathcal{S}$ we will define the action state $\mathcal{A}(s)$, describing all possible action that can be taken by the agent at state s. When taking an action from a state s_t, the agent finds itself in a new state s_{t+1} where it receives a reward $r_{t+1} \in \mathbb{R}$. The action taken is sampled over a probability distribution from the joint space of state and action :

$$\begin{aligned} \pi \ : \mathcal{S} \times \mathcal{A}(s) &\rightarrow [0,1] \\ (s,a) &\rightarrow \pi(s,a) \end{aligned} \tag{2.1}$$

where $\pi(s,a)$ is the probability of picking action a in state s. Such a distribution is called the agent's *policy*. The key goal of reinforcement learning is teaching an agent on how to change its policy to maximize its reward on the long run.

The agent indeed seeks to maximize the *expected return* R_t mapping the reward sequence into $\mathbb{R}$. A commonly used expression for this value employs a *discount factor* $\gamma \in [0,1]$, allowing to make the agent's more sensible to rewards it will get in a close future :

$$R_t = \sum_{i=0}^{T} \gamma^i r_{t+1+i} \tag{2.2}$$

This also allows to adapt this formulation to continuous tasks, where there are no terminal states and the task goes on indefinitely (there are no *episodes* in the learning).

2.1.2 Markov decision process

To make the problem tractable, we ask for the state signal to comply with Markov's property, hence to be *memory-less*. For instance, we want to be able to write that, in a stochastic environment, $\forall s' \in \mathcal{S}$:

$$\mathbb{P}\left(s_{t+1} = s' \,|\, a_t, s_t, \ldots, a_1, s_1\right) = \mathbb{P}\left(s_{t+1} = s' \,|\, s_t, a_t\right) \tag{2.3}$$

Hence, every reinforcement learning problem can be represented by a *Markov Decision Process*, that consists in a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}.(\cdot,\cdot), \mathcal{R}.(\cdot), \gamma)$ where :

- ▷ $\mathcal{S}$ is the agent's state space
- ▷ $\mathcal{A}$ is the agent's action space

- ▷ $\forall s, s' \in \mathcal{S}$, $\forall a \in \mathcal{A}(s)$, $\mathcal{P}_a(s, s') = \mathbb{P}(s_{t+1} = s' \,|\, s_t = s, a_t = a)$ is the probability that action a in state s will lead the agent to transitioning to state s'.
- ▷ $\forall s, s' \in \mathcal{S}$, $\forall a \in \mathcal{A}(s)$, $\mathcal{R}_a(s, s')$ is the immediate reward perceived by the agent when transitioning from state s to s' when taking action a.
- ▷ γ is the discount factor.

A *finite Markov decision process* designates a MDP for which both the action and state space are finite.

2.1.3 State and action value function

Most of the reinforcement learning algorithms are based on evaluation value function. A value function is a function mapping the state space in $\mathbb{R}$, estimating how good (in terms of expected future reward) it is for the agent to be in a given space. More precisely, a value function $V^\pi(\cdot)$ evaluates the expected return of a state when following the policy π. $V^\pi(\cdot)$ is called the **state-value function**.

$$\forall s \in \mathcal{S}, \quad V^\pi(s) = \mathbb{E}_\pi \left[R_t \,|\, s_t = s \right] \tag{2.4}$$

The **action-value function** evaluates the value of taking a given action, and then following the policy π :

$$\forall s, a \in \mathcal{S} \times \mathcal{A}(s), \quad Q^\pi(s, a) = \mathbb{E}_\pi \left[R_t \,|\, s_t = s,\, a_t = a \right] \tag{2.5}$$

Both those functions satisfy particular recursive relationships known as the *Bellman equations*. It is shown that (see TODO quote Sutton) we have the following results :

Bellman equations for Markov Decision Process

- ▷ Bellman equation for the state-value function : $\forall s \in \mathcal{S}$

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s'} \mathcal{P}_a(s, s') \left[\mathcal{R}_a(s, s') + \gamma V^\pi(s') \right] \tag{2.6}$$

- ▷ Bellman equation for the action value function : $\forall s, a \in \mathcal{S} \times \mathcal{A}(s)$:

$$Q^\pi(s, a) = \sum_{s'} \mathcal{P}_a(s, s') \left[\mathcal{R}_a(s, s') + \gamma V^\pi(s') \right] \tag{2.7}$$

2.1.4 Optimal policies

The value functions define a partial ordering in the policy space. A policy π is therefore said to be better than π' (or $\pi \geq \pi'$) if $\forall s \in \mathcal{S}$, $V^\pi(s) \geq V^\pi(s')$. We are looking for π^* so that :

$$\forall \pi, \quad \pi^* \geq \pi \tag{2.8}$$

TODO quote showed that for finite MDPs, there is always at least one policy that is better our equal to all others, and therefore is called the *optimal policy* π^*. As shown in TODO quote, the state-value and action-value function verify the *Bellman optimality equations*.

Bellman optimality equations

- ▷ Bellman optimality equation for the state-value function : $\forall s \in \mathcal{S}$

$$V^\pi(s) = \max_{a \in \mathcal{A}(s)} \pi(s, a) \sum_{s'} \mathcal{P}_a(s, s') \left[\mathcal{R}_a(s, s') + \gamma V^\pi(s') \right] \tag{2.9}$$

- ▷ Bellman optimality equation for the action value function : $\forall s, a \in \mathcal{S} \times \mathcal{A}(s)$:

$$Q^\pi(s, a) = \sum_{s'} \mathcal{P}_a(s, s') \left[\mathcal{R}_a(s, s') + \max_{a \in \mathcal{A}(s')} Q(s', a') \right] \tag{2.10}$$

Those relations are essential in understanding the solving algorithms that will be presented later.

There exists several ways of solving (i.e computing the optimal policy) of a Markov Decision Process, that can generically be separated in two categories : *model-based* and *model-free* methods.

2.2 Dynamic Programing

Dynamic programing is a mathematically well-developed theory. It requires the complete and accurate model of the environment, making it a model-based method.

Dynamic programing methods aims at computing the optimal value function at every state of state space. This could, of course, be done by solving the $|\mathcal{S}|$ equations of $|\mathcal{S}|$ unknowns that are the Bellman equations for a given policy, and then evolve that policy toward a better one, based on the current value function. Of course, this approach is computationally intractable for large state space and therefore needs to be adapted, but gives a first approach of the idea behind dynamic programing.

2.2.1 Generalized policy iteration

The generalized policy iteration methods rely on alternating two processes, known as policy evaluation and policy improvement.

- ▷ Policy evaluation deals with estimating the value function of a given policy π, without directly solving the full system given by Bellman equations. The idea is actually pretty simple : use Bellman's equation as an update rule, using that the value function is a fixed point for this update rule. The algorithm, after setting the tabled value function to an initial value, iterate by performing what is called *full Bellman backups* :

$$\forall s \in \mathcal{S}, \quad V^{\pi}_{k+1}(s) = \sum_{a \in \mathcal{S}} \pi(s,a) \sum_{s'} \mathcal{P}_a(s,s') \left[\mathcal{R}_a(s,s') + V^{\pi}_k(s')\right] \tag{2.11}$$

  This algorithm converges under the same assumptions that guarantee the existence of the value function, and has the generic name of *iterative policy evaluation*. They are many refining for speeding up that process (reduced backups, prioritized sweeping) that we won't address here.

- ▷ Policy evaluation is a process that from a given policy value function, returns a better or equal policy compared to the latter. The simplest way to do that is for every state $s \in \mathcal{S}$, consider every action-value functions :

$$Q(s,a) = \sum_{s'} \mathcal{P}_a(s,s') \left[\mathcal{R}_a(s,s') + \gamma V\pi(s')\right], \quad a \in \mathcal{A}(s) \tag{2.12}$$

  and to build π' to be *greedy* with respect to those actions-values :

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \{Q(s,a)\} \tag{2.13}$$

  The policy improvement theorems then ensures that $\pi' \geq \pi$.

Hence, generalized policy improvement are a set of methods that iteratively combine those two sub-methods to compute the optimal policy for a given MDP. Of course, one does not have to perform all sweeps of value evaluation before improving the policy to converge toward an optima (indeed, many times our sweeps won't have any affect on the greedy policy). They are many ways to combine the two (prioritized sweeping, asynchronous dynamic programing), but the most used and one of the most quickest way to converge is to use the value iteration algorithm.

2.2.2 The value iteration algorithm

The value iteration algorithm takes the limit of the behavior we just described, and stops the value evaluation procedure after only *one state space sweep*. It therefore performs a simple backup procedure :

$$\forall s \in \mathcal{S}, \quad V_{k+1} = \max_{a \in \mathcal{A}} \sum_{s'} \mathcal{P}_a(s, s') \left[\mathcal{R}_a(s, s') + \gamma V_k^{\pi}(s')\right] \tag{2.14}$$

For any arbitrary V_0, it is shown that $V_k \to V^*$ as $k \to \infty$, under the same hypothesis that ensure the existence of the optimal value function V^*. As one can notice, it actually implements the *Bellman optimality conditions* as an update rule !

2.3 Temporal differences methods

Temporal difference methods can be seen as a combination of dynamic programing and another kind of learning called Monte Carlo methods, where the expected return are approximated via sampling. Like dynamic programming, TD methods are said to bootstrap (meaning that they build their estimators through already estimated values), but are *model-free* methods and learn from experience.

The justification, proof of convergences and literature and those models is pretty wide, hence we will not cover them in this report. However, full description of those methods can be found in TODO quote.

2.3.1 On-policy method : SARSA

The SARSA algorithm is an *on-policy* control method, meaning that the algorithm updates the value function and improves the current policy it is following. At state s_t, it chooses an action a_t from its policy and follows it. After observing the reward r_{t+1} and the next state s_{t+1}, it again chooses an action a_{t+1} using a soft policy and performs a one-step backup :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \tag{2.15}$$

It therefore relies on a 5-tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ to perform the udpate, giving it the State Action Reward State Action (SARSA) name.

The General Sarsa Algorithm

1. *Initialize* $Q(s, a)$ arbitrarily $\forall (s, a) \in \mathcal{S} \times \mathcal{A}(s)$
2. *Repeat* for each episode :
   Initialize s
   Choose $a \in \mathcal{A}(s)$ using a soft policy derived from Q (typically ε-greedy)
   Repeat for each step of the current episode :
   Take a, observe r, s'
   Choose a' from s' using policy derived from Q
   $Q(s, a) \longleftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$
   $a \leftarrow a'$, $s \leftarrow s'$

until $s \in \mathcal{S}^+$.

The convergence properties of SARSA depend on the nature of the policy's dependency on Q. Indeed, SARSA converges with probability 1 to the optimal policy as long as all the sate and actions pairs are visited an infinite number of time, and the policy converges in the limit to the greedy policy. This is done, for instance, by turning the temperate of a softmax based policy to 0, or by having $\varepsilon \to 0$ for a ε-greedy policy. For SARSA to converge, we also as the learning rate to comply with the stochastic approximation conditions :

$$\sum_k \alpha_k(a) = +\infty \quad \textit{and} \quad \sum_k \alpha_k(a)^2 < +\infty \tag{2.16}$$

where $\alpha_k(a)$ is the learning rate for the k^{th} visit of the pair (s, a).

2.3.2 Off-policy method : Q-learning

The Q-learning algorithm is an off-policy method who learns to directly approximate Q^*, independently of the policy being followed. Its update rule is given by :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.17}$$

The actual policy being followed still has an effect in that it determines which state-actions pairs are visited and updated. However all that is required for convergence it that all pairs continue to be updated.

Q-Learning Algorithm

1. *Initialize* $Q(s, a)$ arbitrarily $\forall (s, a) \in \mathcal{S} \times \mathcal{A}(s)$
2. *Repeat* for each episode :
   Initialize s
   Repeat for each step of the current episode :
   Choose $a \in \mathcal{A}(s)$ using arbitrary policy
   Take a, observe r, s'
   $Q(s, a) \longleftarrow Q(s, a) + \alpha \big[r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s', a') - Q(s, a) \big]$
   $s \leftarrow s'$

until $s \in \mathcal{S}^+$.

Along with this hypothesis and a slight variation in the usual stochastic approximation conditions, the learned action value function by Q-learning has been shown to converge to Q^* with probability 1.

2.3.3 Eligibility traces

In TD(0) approach (described in the latest section), we update the value function in the direction of the *one-step return* :

$$\Delta V_t(s_t)^{(1)} = \alpha \left[r_t + \gamma V_t(s_{t+1}) - V_(s - t) \right] \tag{2.18}$$

The idea behind eligibility traces is to expand that update rule in order to steer the value fonction towards the *n-step return* (or at least until a terminal state is reached) :

$$\begin{aligned} \Delta V_t(s_t)^{(n)} &= \alpha \left[r_t + \gamma r_{t+1} + \ldots + \gamma^n V_t(s_{t+n}) - V_t(s_t) \right] \\ &= \alpha \left[R_t^{(n)} - V_t(s_t) \right] \end{aligned} \tag{2.19}$$

The backups can not only be done toward any n-step return, but toward any average of such returns, as long as the corresponding weights sum-up to one. In this way, the TD(λ) algorithm can be understood as a particular way of averaging n-steps returns. With $\lambda < 1$, the resulting backup is known as the λ-*return* :

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \tag{2.20}$$

where the weights are fading with n. When the runs are episodic, we can write this return as :

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-} R_t(n) + \lambda_{T-t-1} R_t^{(T)} \tag{2.21}$$

Sarsa(λ) algorithm

1. *Initialize* $Q(s,a)$ arbitrarily $\forall (s,a) \in \mathcal{S} \times \mathcal{A}(s)$
2. *Repeat* (for each episode) :
   Initialize s, a
   Repeat for each step of the current episode :
   Take action a, observe r, s'.
   Choose $a' \in \mathcal{A}(s')$ using soft policy derived from Q
   $\delta \longleftarrow r + \gamma Q(s', a') - Q(s,a)$
   $e(s,a) \longleftarrow e(s,a) + 1$
   For all s, a :
   $Q(s,a) \longleftarrow Q(s,a) + \alpha \delta e(s,a)$
   $e(s,a) \longleftarrow \gamma \lambda e(s,a)$
   $s \leftarrow s'$, $a \leftarrow a'$
   until $s \in \mathcal{S}^+$.

Such a formulation of eligibility traces is known as the *forward view* of TD(λ), and shows how eligibility traces build the bridge between TD(0) methods and Monte-Carlo one. It is not implementable as is since it is non-causal. There exist a more mechanistic view, equivalent to the forward view (see [SB98]), known as the *backward-view*. It gives birth to causal version of the TD(λ) method. We give as an example the pseudo-code for the SARSA(λ) algorithm above. Its Q-learning equivalent can be found in [SB98].

2.4 Grid-world examples

We hereinafter describe two grid-world state spaces, on which we will apply the learning rule derived in the latest section. Such example are trivial and are displayed here just to show convergence and behavior of the different algorithms.

We'll consider the two following state spaces :

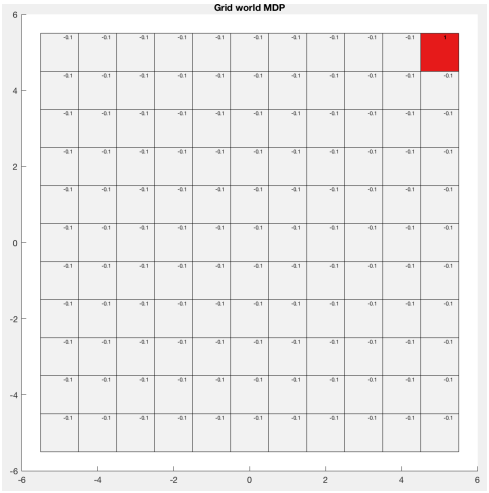


Figure 2.1: The *free_grid* state space

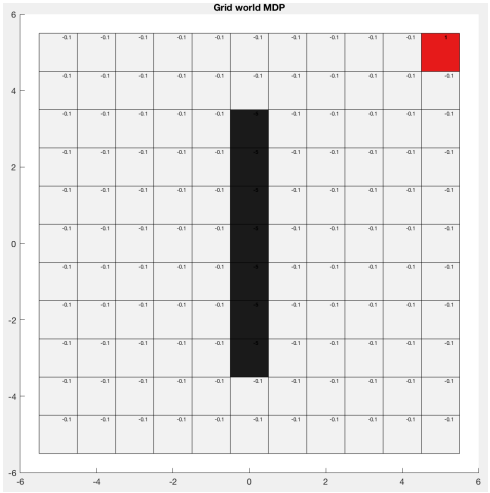


Figure 2.2: The *bar_grid* state space

2.4.1 Dynamic Programing solving

Let us run the DP algorithm on such grid worlds. We'll consider a stochastic environnement, with the transition probability :

$$\mathcal{P}^a_{s,s'} = \begin{cases} 0.9 & \text{if } s' = a(s) \\ 0.1 & \text{otherwise} \end{cases} \tag{2.22}$$

Running the value iteration algorithm (assuming we now the environment model), we obtain the following policies and learning curves. The stopping criterion adresses the maximum absolute change brought to the value function as the sweeping goes through the state space :

$$\text{If } \max_{s \in \mathcal{S}} |V_{k+1}(s) - V_k| < \delta \text{ then stop} \tag{2.23}$$

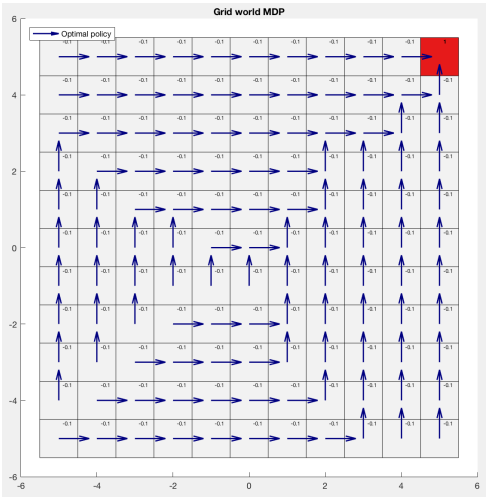


Figure 2.3: The *free_grid* learned optimal policy

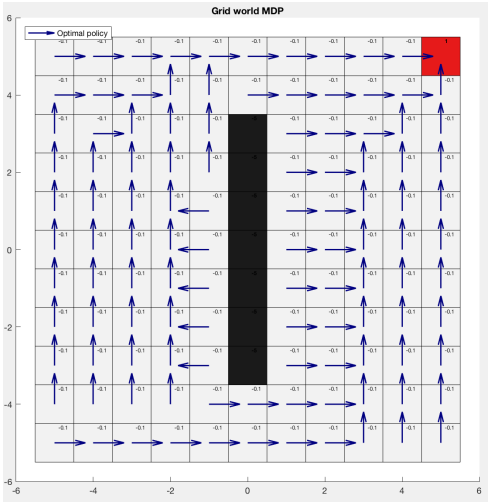


Figure 2.4: The *bar_grid* learned optimal policy

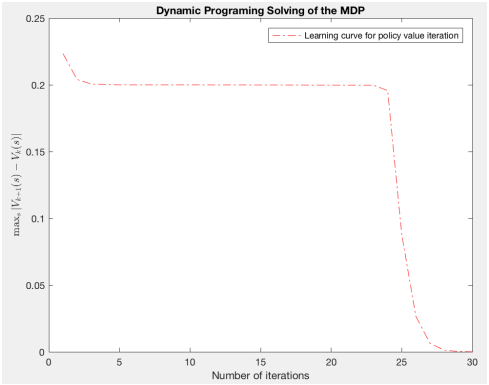


Figure 2.5: The *free_grid* value iteration learning curve

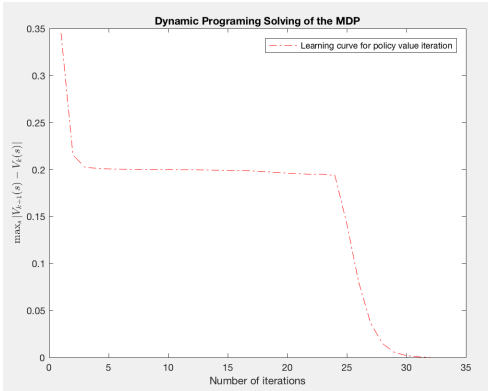


Figure 2.6: The *bar_grid* value iteration learning curve

One can notice that for the *bar_grid* environment, the agent undergoes a longer trajectory than necessary, at the left of the obstacle. This is because of the stochastic nature of the environment, causing the agent to learn to takes its distance with the obstacle in order not to accidentally hit it (and then receive a very negative reward).

The learned policy are indeed optimal. The next algorithms (SARSA and Q-learning) will try to reproduce them without a model for the environment.

2.4.2 SARSA solving

We display here the learning curves for the *free_grid* state space using SARSA. The algorithm manages to learn the optimal policy and the right action-value functions. We use *optimistic initialization* to

encourage exploration, and Gibbs sampling for following a soft-policy : $\forall(s,a) \in \mathcal{S} \times \mathcal{A}(s)$

$$\pi(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in \mathcal{A}(s)} e^{Q(s,a')/\tau}} \tag{2.24}$$

We'll tune the distribution's temperature τ to zero, in order to converge toward the greedy policy w.r.t the learned action-value function.

Following this strategy and tuning our learning rate to comply with the stochastic approximation conditions, we obtain the following learning curves. Again, our stopping criterion addresses the *maximum change in the acton-value function over all the trajectories of a mini-batch.*

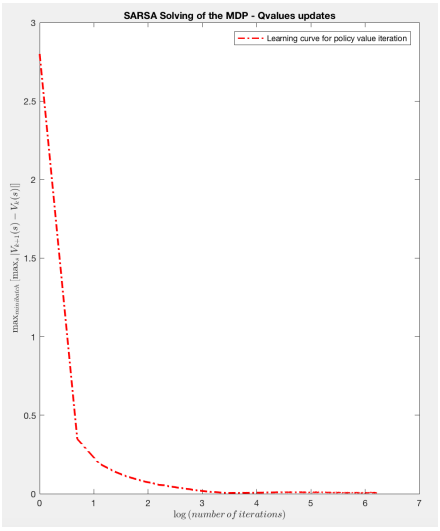


Figure 2.7: Learning curve for SARSA on *free_grid*

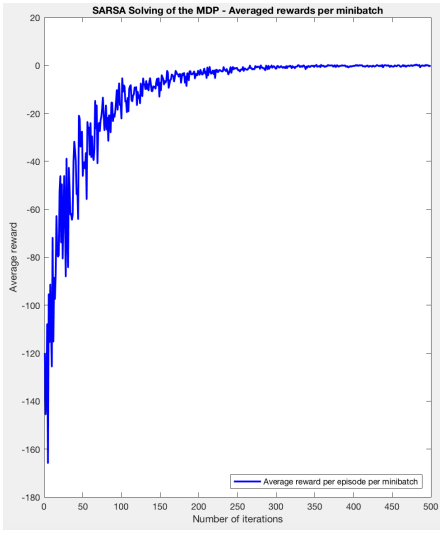


Figure 2.8: Averaged rewards over mini-batch for SARSA on *free_grid*

2.4.3 Q-learning solving

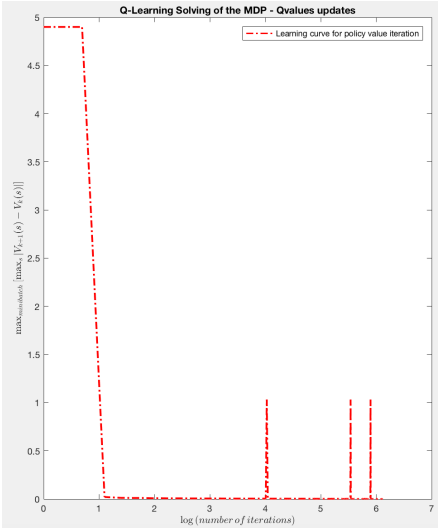


Figure 2.9: Learning curve for Q-learning on *bar_grid*

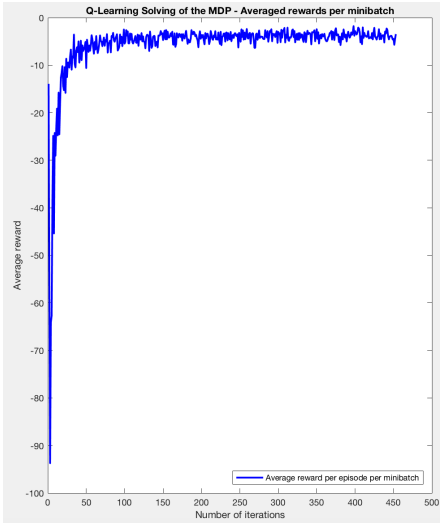


Figure 2.10: Averaged rewards over mini-batch for Q-learning on *bar_grid*

We display here the learning curves for the *bar_ grid* state space using Q-learning. Again, the algorithm manages to learn the optimal policy and the right action-value functions. We use Gibbs sampling for the behavior policy, without any tuning for the temperature (the behavior policy doesn't need to be greedy in limit). The learning curves obtained are given hereinafter.

Chapter 3

Compliant Reinforcement Learning

With our approach, we wish to tackle two topics : on one hand, we wish to develop an imitation based learning framework that *accelerates* the learning process. Also, we wish to apply a shifting compliance based behavior so that an agent can overcome an arbitrarily suboptimal teacher.

3.1 Principle

We start by making a fairly strong hypothesis to simplify our reasoning. In the following chapters, we will consider that a mentor demonstration provides one recommended action for every state - which is the equivalent of providing one deterministic policy. This somehow out-scopes the case of unique demonstration, but can be understood as a combination of multiple demonstrations.
Hence, we will consider that a teacher's demonstration is a mapping between the state space $\mathcal{S}$ of the learner and its action set $\mathcal{A}(s)$, $\forall s \in \mathcal{S}$, denoted π_m :

$$\begin{aligned} \pi_m \ : \mathcal{S} &\rightarrow \mathcal{A} \\ s &\rightarrow a_m(s) \end{aligned} \tag{3.1}$$

where $a_m(s)$ is the *recommended action* of the mentor at state s.
Such an hypothesis isn't trivial, and will be discussed later in this report. The main reason it is considered is that it enables to treat the whole state space in the same way - without having to distinguish regions in the state space that are provided with demonstrations and regions which are not.

As discussed earlier in this report, our goal is to mimic the shifting compliance a child can have with respect to its teacher when learning a new skill. Because this implies making choices as to wether follow a recommended action or sample elsewhere in the action space, it is clear that only the action selection (hence the *exploratory policy*) should be impacted by the presence of the mentor.

We will now consider an action selection process based on the teacher's recommandation. We introduce a parameter p, that can be understood as a **confidence measure** in the teacher. The action selection process we chose to follow can be understood as a p-greedy action selection with respect to the teacher recommandation and is defined as :

$$\forall s \in \mathcal{S}, \quad \pi(s) = \begin{cases} a_m \text{ with probability } p \\ a \in \mathcal{A}(s) \text{ with probability } (1-p) \end{cases} \tag{3.2}$$

The learner therefore has two possibilities : follow the teacher with probability p, or take its own action, with probability $(1-p)$. This motivates to call p a confidence measure : the greater p is, the more the learner will trust the teacher and follow its recommandation. In the case where the learner decides to take its own action, it samples in its state space through Gibbs softmax sampling, thanks to its current action-value estimates.

The purpose of sections (3.4) and (3.5) is to provide p with different dynamics through time and evaluate the corresponding performances.

The updates will follow the SARSA algorithm, and the usual TD(0) updates : $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + Q(s',a') - Q(s,a) \right] \tag{3.3}$$

which are indeed not impacted by the presence of a mentor. Therefore, under the very simple conditions for SARSA to converge to a locally optimal policy (that is, among others, that the exploratory policy is greedy in limit), our algorithms will converge too.

3.2 Experimental MDP

To test the effectiveness of the solutions we willl propose, we decided to provide ourself with a model that would stay fixed all along the experiments, in order to compare the different algorithms we could come up with.

We designed the state space displayed in figure (3.1). In this environment, all black cells are obstacles. They give out highly negative rewards ($r = -10$). Whenever an agent take the action to enter such a cell, it immediately perceives the negative reward but stays in its current cell. The only positive reward is at the middle of the grid ($r = 10$), the only terminal state. Any episode starts at one of the corner of the grid (green cells), and every step spent on a non-terminal cell gives out a small negative reward ($r = -0.1$). The transitions are stochastic, with the transition probability model :

$$s' = \begin{cases} a(s) \text{ with proba } 0.95 \\ s'' \neq a(s) \text{ otherwise, uniformly sampled} \end{cases} \tag{3.4}$$

This state space is big enough for the usual algorithms to learn rather slowly, even if they are greatly enhanced by the use of eligibility traces. Also, all tested algorithms (SARSA, Q-learning, SARSA(λ) and Watkins Q(λ)), because they do not perform infinite exploitation / exploration moves, renders slightly suboptimal policies.

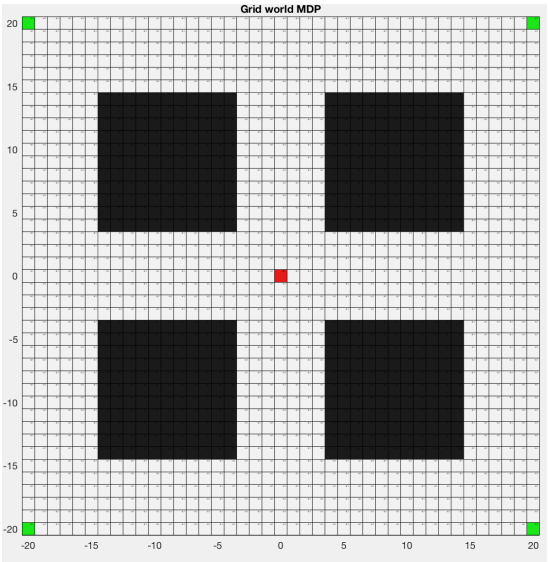


Figure 3.1: The *maze_grid* state space

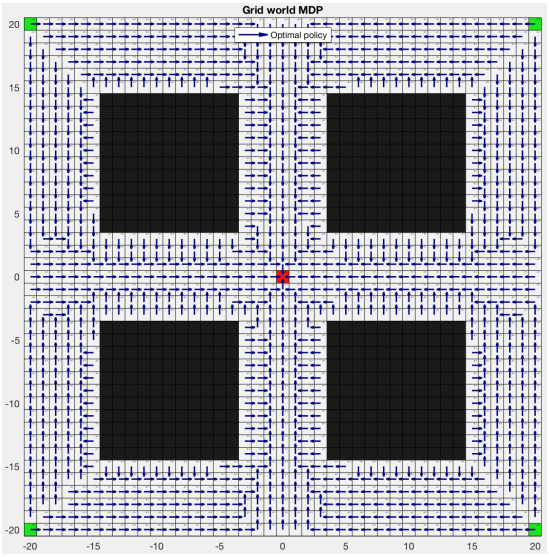


Figure 3.2: Optimal policy, computed with DP

If we go back at one of our motivating example (robot grasping an object), we could easily draw parallels between such an example and the grid environment we just presented. Indeed, we could imagine a teacher providing a demonstration that borders the obstacle. Because the learner suffers a (slightly) stochastic dynamic, this would indeed be a largely suboptimal solution, since large negative rewards will be likely to occur during the learning. However, the demonstration contains a fairly important information, that is the direction to follow to reach the center of the grid.

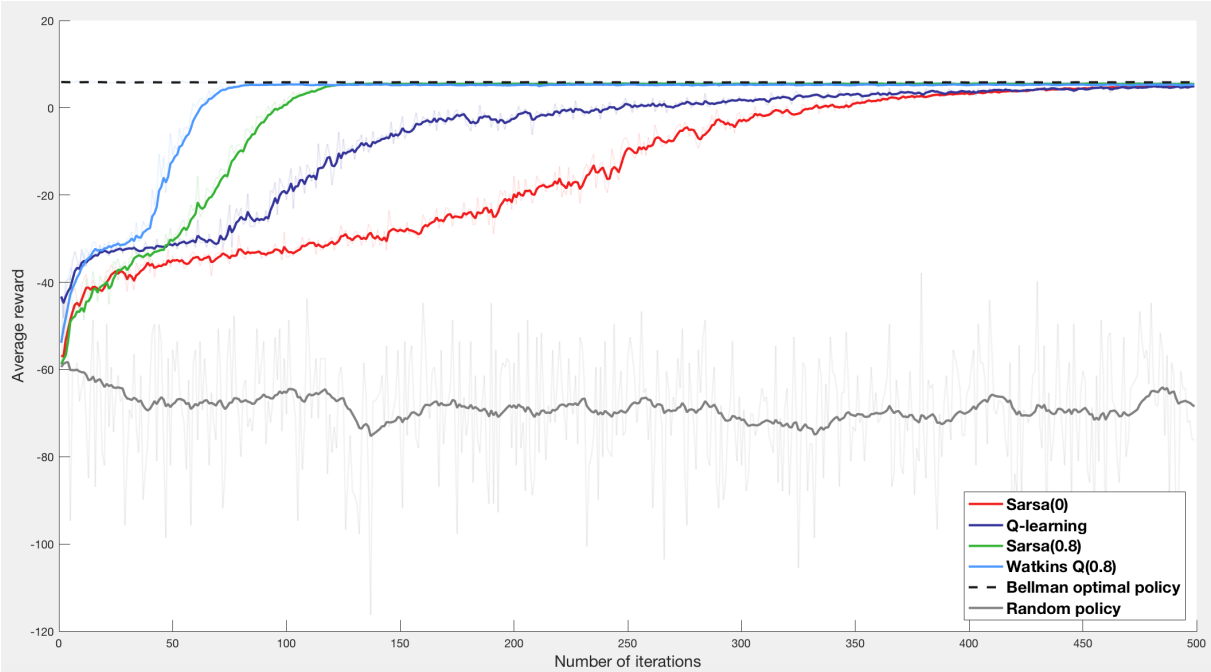


Figure 3.3: Average rewards on minibatch for learned policies, optimal policy and random policy.

Figure (3.3) displays the convergence (expressed as average reward on minibatch) for Qlearning, Sarsa(λ), Sarsa(0) and Watkins Q(λ). By average reward on minibatch, we mean that at every iteration, the reward is average around a given number of trajectories, following the same *exploratory* policy. This allows to reduce the stochasticity of trajectories while learning and give a smoother estimation of how well the algorithm is learning. As a way of comparing them to the optimal and random policies, we also plot the average rewards perceived by the latest along many trajectories.
For all the learning algorithms tested, we used optimistic action-value initialisation to promote exploration. This explains why many negative rewards are perceived in the beginning.

3.3 Generating mentors

More than giving us an idea how generic reinforcement learning algorithms performs on our sandbox MDP, coding those different methods enables us to *generate mentors of varying sub-optimal levels.*

If we consider a SARSA learner (for instance), we can at any time of its learning generate a deterministic version of the current exploratory policy. This is done by taking its greedy version with respect to its current Q-values estimates. Hence $\forall s \in \mathcal{S}$:

$$\pi_m(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \{Q(s,a)\} \tag{3.5}$$

Such a process is displayed in figure (3.4). Because it is now longer stochastic, the mentor is slightly better than the learner it was generated from, but still is clearly suboptimal. This method hence enables us to generate mentor of different optimality levels.

3.4 Naive learners

3.4.1 Constant compliance

In the context of the action-selection described in (3.2), we decided to first implement a fairly naive method. It consists in following the teacher's recommandation with a **constant** confidence term p.

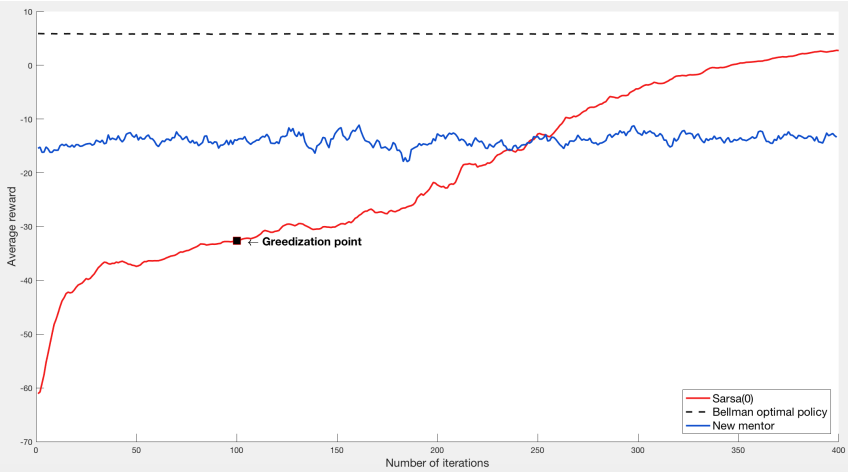


Figure 3.4: Generating a suboptimal mentor from a SARSA learner

Therefore, the learner *complies* with the teacher with probability p, and decide to choose its own action (that **could include** a_m) with probability $1 - p$. The softmax sampling used when discarding a recommandation is tuned by a decaying temperature coefficient, making it greedy in limit.

The figures (3.5) and (3.6) display the learning obtained (cumulated rewards averaged over mini-batches) when fed with, respectively, the optimal policy for the MDP and a slightly suboptimal one.

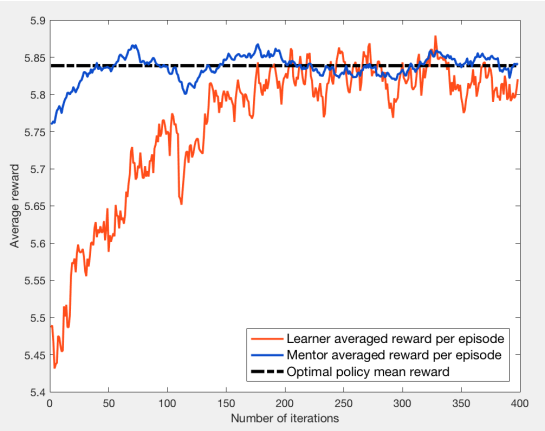


Figure 3.5: Constant compliance learning, $p = 0.9$, with the optimal mentor

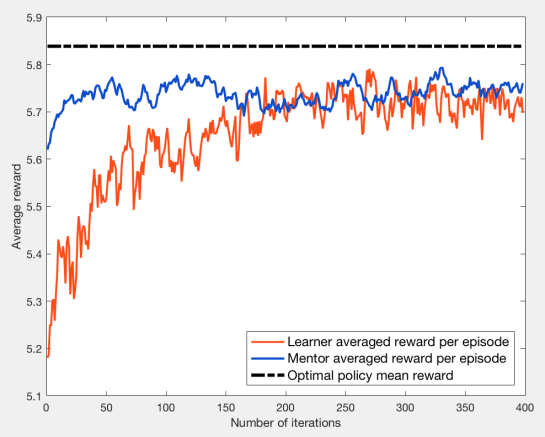


Figure 3.6: Constant compliance learning, $p = 0.9$, with a slightly suboptimal mentor

Figure (3.5) shows that from its exploration, the learner is able to quickly learn his way thanks to the optimal mentor. As expected, it eventually follows the mentor's action, wether it complies or not, since the recommended action holds the best action-values. On the other hand, as shown in figure (3.6), the high confidence the learner initially have in its mentor prevents it from reaching optimal performance, and the policy its renders actually mimic its mentor suboptimal one. Still, one could expect the learned policy to be slightly better than its suboptimal teacher's one, even if not optimal. However, achieving this comes with a lot of effort in the tuning of p and of the softmax distribution temperature decrease coefficient.

This remark actually pinpoints a major downside of this method, that is the need of fine tuning of the parameter p. But obviously, there is even a bigger downside, that becomes a major game killer when dealing with largely suboptimal mentors. Indeed, some mentors can be suboptimal enough to only show a good direction of exploitation, but not be able to reach the target. . Figure (3.7) displays such a policy,

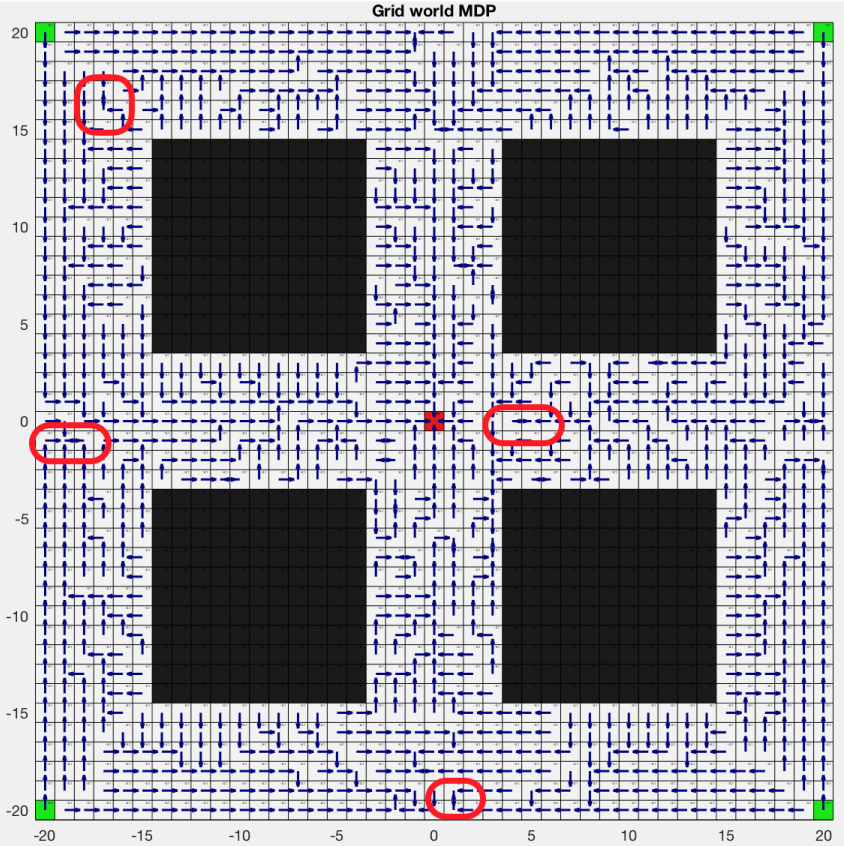


Figure 3.7: An exemple of suboptimal mentor policy that doesn't always lead to the positive reward

where the mentor's policy creates loops and does not always leads to the positive reward.
. Applying the latest method with such a teacher is fatal for the learning, as the learner discovers that following the mentor's action yields largely negative rewards, but is not able to bypass them as it keeps a constant confidence in its teacher. This leads the learner to build up low Q-values in the directions recommended by the mentor, and to try to follow an opposite path. This is highly counter-productive since the mentor still gives the right exploration direction.

3.4.2 Vanishing compliance

One of the downside of the constant compliance approach is that the exploratory behavior is always biased by the mentor's recommandation. This breaks the need of this policy to be *greedy in limit* (which is a specification for SARSA algorithm to converge). In the case of a sub-obptimal mentor, this means that the optimal behavior could never be reached.

We now decide to comply with the need to be greedy in limit. Therefore, we decide to set p to be *constantly decreasing along the learning procedure* :

$$p_{t+1} = \beta p_t \tag{3.6}$$

with $\beta < 1$. As before, the temperature of the Gibb's softmax sampling also goes to 0 as the learning goes on.

The action-selection is therefore biased by the mentor's recommandation in the beginning of the learning only, and slowly decides to take its own choices, based on its current Q-values estimates. This approach sounds more promising as the learner is more likely to quickly discover the location of high rewards (following the teacher policy with p close to 1), and will then makes it own exploration along those trajectory, to end up in a setting where the teacher's actions are now longer considered.

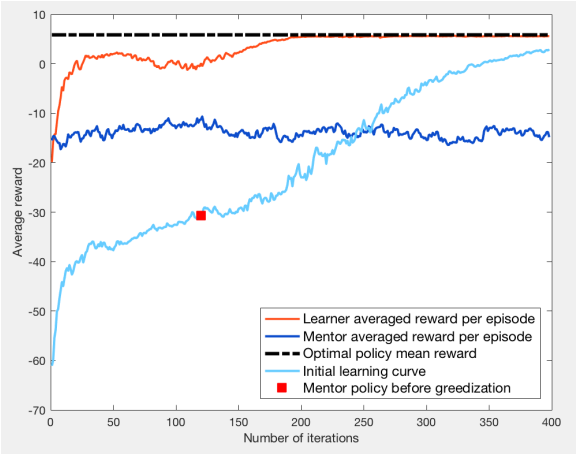


Figure 3.8: Vanishing compliance, $\beta = 0.99$

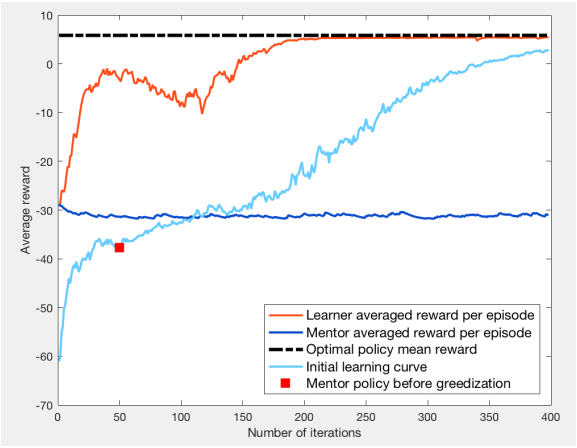


Figure 3.9: Vanishing compliance, $\beta = 0.97$

Figures (3.8) and (3.9) display the result on two sub-optimal policies, that were derived from the Q-values learned at a given moment (denoted through a red square) in a learning process. They show that with this approach, while the learner is compliant with the teacher, it steadily increases its accumulated reward thanks to its exploratory actions. However, it then comes to a plateau (or even an undershoot) when the guidance by the teacher becomes too weak as p reaches a critical level. The learner then mostly relies on its exploratory actions (which volatility are guided by its temperature coefficient) to explore new regions of the state space. Once it has somehow learnt the action value function related to this part of its state space, and as its temperature goes down, the learner's accumulated reward goes up until it reaches the level of the optimal policy.

If this approach seems to work relatively well, there are still some critical downsides to it. First of all, a lot of tuning is required in order to find the hyper-parameters (p_0, β, temperature evolution, ..) that lead to a fast learning. Also, there is an undershoot in the learning that seems to slow it down, due to the exploring phase that happens when p becomes too low. Such exploration could be avoided or reduced if the learner is able to figure out quickly that indeed, the mentor was right in its recommandation.

3.5 Adaptative compliance learners

Up to this point, it makes sense to somehow *learn* the optimal value of the confidence parameter p, so that we don't have to manually tune its evolution, and so that it can store how right the teacher recommandations are. Ideally, we would like p to be near 0 when the teacher provide a suboptimal action and 1 when the recommended action is sampled from the optimal policy. The underlying problem is therefore to infer the optimality of the teacher in different regions of the state space.

We hereinafter describe two methods which attempt to do so. Their respective experimental results as well as further discussion will be found in chapter (4).

3.5.1 Implicit β-compliance

Let us define the *confidence* term p locally for every state of the state space. For each $s \in \mathcal{S}$, $p(s)$ is given a Beta prior : $p(s) \sim \beta(\alpha(s), \beta(s))$ and represents the initial trust we have in a mentor's recommandation at state s. The initial values α_0 and β_0 define the initial prior belief we have over p (for instance, $\alpha = 0.5$ and $\beta = 0.055$ define a prior belief that the teacher is most probably right).

As in before, we perform a *p-greedy policy* with respect to the teacher recommandation :

$$\forall s \in \mathcal{S}, \quad \pi_p(s) = \begin{cases} a_m \text{ with probability } p(s) \\ a \in \mathcal{A}(s) \text{ with probability } (1 - p(s)) \end{cases} \tag{3.7}$$

The following describe how we wish to learn the value of $p(s), \forall s \in \mathcal{S}$. Based on the 5-tuple (s, a, r, s', a') obtained thanks to the action selection process we just described, we compute at every step the following temporal difference value :

$$\delta_t = r + \gamma Q(s', a') - Q(s, a_m) \tag{3.8}$$

which compares, *in average*, the advantage (our drawback) of following the state-action pair (s, a) rather than the one indicated by the teacher, according to the current Q-values estimates.

We then apply the following update rule to $p(s)$:

$$\begin{aligned} \alpha_t(s) &\leftarrow \alpha_t(s) + \mathbb{1}_{a=a_m} \delta_t \varepsilon_t \\ \beta_t(s) &\leftarrow \beta_t(s) + \mathbb{1}_{a \neq a_m} \delta_t \varepsilon_t \end{aligned} \tag{3.9}$$

The intuition behind this update rule is simple : if we see that the expected return for the mentor action increases (resp. decreases), then we increase (resp. decrease) α which results in a shift of p toward a larger (resp. smaller) confidence. A similar reasoning holds for the β term. ε_t is the update rule's learning rate, which value and dynamic will be discussed later.

3.5.2 Explicit compliance

We consider here a somewhat similar approach, where our listen versus discard exploration policy is computed according to the current estimated values of the actions *'listen'* and *'discard'*. Let us introduce the action spaces :

$$\forall s \in \mathcal{S},\ \mathcal{A}_c(s) = \{'listen', 'discard'\} \tag{3.10}$$

to which we assign the action values $Q_c(\cdot, l)$ and $Q_c(\cdot, d)$ (where $'l'$ denotes the action of listening and $'d'$ the action of discarding the teacher recommendation).

The exploration is done by computing a soft policy derived from $\{Q_c(s, l), Q_c(s, d)\}$ for all $s \in \mathcal{S}$. We do this using a Gibbs softmax distribution, which yields :

$$\forall s \in \mathcal{S}, \quad \pi_c(s) = \begin{cases} 'l' \text{ with probability } p = \sigma\left(\dfrac{Q_c(s, l) - Q_c(s, d)}{\tau}\right) \\ 'd' \text{ with probability } 1 - p \end{cases} \tag{3.11}$$

where $\sigma(\cdot)$ is the logistic sigmoid and τ is a temperate coefficient decreasing to 0 (greedy policy in limit).

After each SARSA update, we also make the following update :

$$\begin{cases} Q_c(s, l) \leftarrow \beta Q_c(s, l) + (1 - \beta) Q(s, a_m) \\ Q_c(s, d) = \beta Q_c(s, d) + (1 - \beta) \max\limits_{a \neq a_m} Q(s, a) \end{cases} \tag{3.12}$$

with $\beta > 0$ the update rule's learning rate.

We can introduce some prior confidence in the teacher by setting, $\forall s \in \mathcal{S}$:

$$Q_c^0(s, l) - Q_c^0(s, d) = \log\{\frac{p}{1 - p}\} \tag{3.13}$$

where p is the retained probability of initially choosing to listen to the teacher.

Chapter 4

Results

4.1 Implicit β-compliance

We hereinafter give some practical considerations for the implicit β-compliance and explicit compliance methods, and discuss the experimental results we obtained.

4.1.1 Practical considerations

One of the main specification of the SARSA learning paradigm is that the exploration policy must be *greedy in limit* so that a fixed point can emerge (hopefully the set of Q-values related to the optimal policy). In an actor-critic approach, the usual way to bring the *critic* term in a stationary regime is to modify its learning rate to take the probability of taking an action into account (see [SB98]).
This is the approach we chose in order to pick the learning rate ε_t of the update rule (3.9), which now becomes :

$$\begin{aligned} \alpha_t(s) &\leftarrow \alpha_t(s) + \mathbb{1}_{a=a_m}\gamma\delta_t p(s) \\ \beta_t(s) &\leftarrow \beta_t(s) + \mathbb{1}_{a\neq a_m}\gamma\delta_t(1-p(s)) \end{aligned} \tag{4.1}$$

with $\gamma > 0$. This update rule can still be simplified, by approximating p by its mean value, which gives the update rule that we applied in practice :

$$\boxed{\begin{aligned} \alpha_t(s) &\leftarrow \alpha_t(s) + \mathbb{1}_{a=a_m}\frac{\gamma\alpha_t(s)}{\alpha_t(s)+\beta_t(s)}\delta_t \\ \beta_t(s) &\leftarrow \beta_t(s) + \mathbb{1}_{a\neq a_m}\frac{\gamma\beta_t(s)}{\alpha_t(s)+\beta_t(s)}\delta_t \end{aligned}} \tag{4.2}$$

This ensures us that we will direct the update toward a fixed point, and end up with a greedy policy (in limit).

In this method, p is given a Beta prior distribution. To perform the action selection, we would therefore need to sample $p(s)$ from its current distribution, and then sample a Bernoulli random variable of parameter $p(s)$. Assuming that the Beta distribution is sharply peaked around its mean (which we will guarantee by choosing an appropriate prior and learning rate γ). Then $p(s)$ can be approximated by its mean value $\mathbb{E}[p(s)] = \dfrac{\alpha(s)}{\alpha(s)+\beta(s)}$. We therefore only need to sample a Bernoulli random variable of parameter $\mathbb{E}[p(s)]$ at every state to complete the action selection process.

The tuning that needs to be done is therefore left to γ, the prior and the temperature. In this framework, tuning is made much simpler and one only has to check that the updates are of the same order of magnitude with the prior (in order for the posterior distribution to adapt to observations, but also to retain the memory of the prior).

4.1.2 Results

Figures (4.1) and (4.2) show the learning curves for an agent using the β-compliance, with two different suboptimal teachers. As before, both of them were generated by observing the Q-values of a "normal" learner (without mentor) and by creating the greedy policy with respect to them.

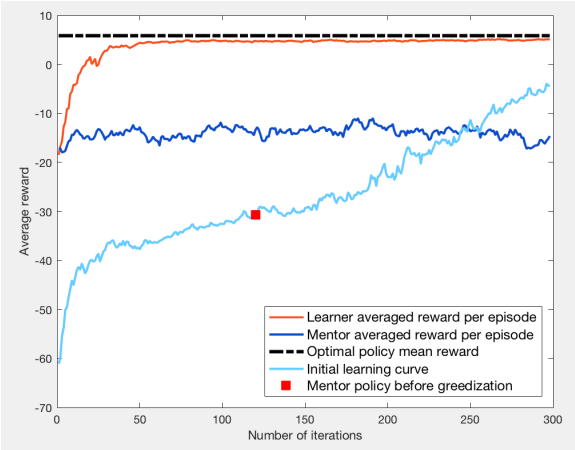


Figure 4.1: Actor-critic learnt compliance learning curve for

Figure 4.2: Actor-critic learnt compliance learning curve

The first remark we can make is that by learning the confidence, we were able to avoid the undershoot that we observed for the vanishing compliance method (see figures (3.8) and (3.9)). This results in a faster learning and better behaving learning curves. The different methods will be further compared in (4.3).

4.1.3 Discussion

It would be interesting to see what is the *posterior distribution* of the $p(s)$, $s \in \mathcal{S}$. We expect our final policy to be greedy, and hence the posterior distribution of the compliance term to be sharply peaked around 0 or 1, with proportion given by how good the mentor is.. Figures (4.3) and (4.4) display the histogram of the repartition of the mean of $p(s)$, $s \in \mathcal{S}$ (again, we approximate the Beta distribution by its mean to have an understandable visualization), for two different mentors (figure (4.3) uses the mentor of (4.1) and figure (4.4) the mentor of (4.2)) . As expected, most of the means are either close to 0 and 1, and they are more means close to 0 (poor confidence) when using the second mentor that is far worse than the first.

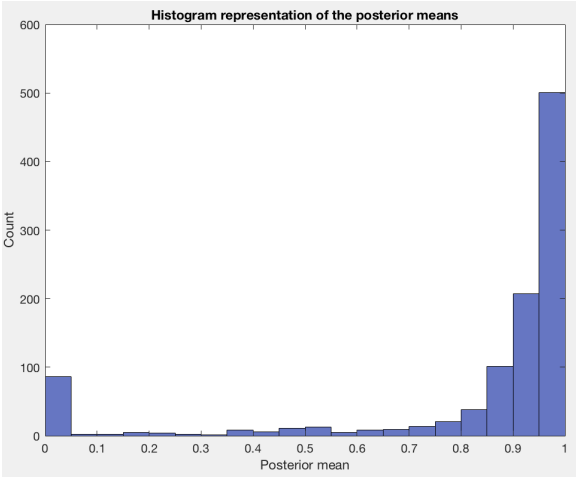


Figure 4.3: Histogram representation of the posterior means for a suboptimal teacher

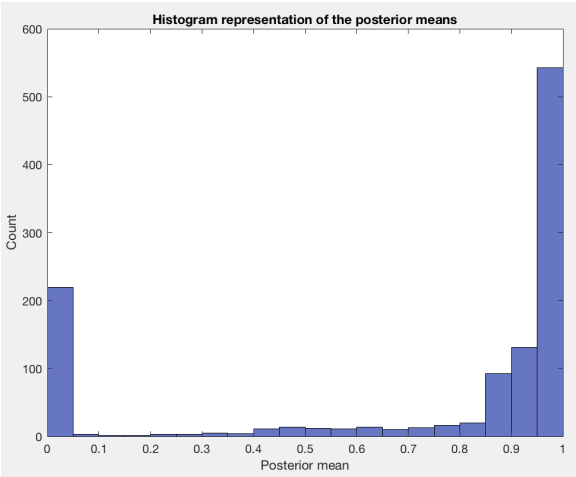


Figure 4.4: Histogram representation of the posterior means for a even more suboptimal teacher

Also, to better understand how this algorithm works, it would be interesting to visualize the areas where the learner rejects or listen to its mentor. We expect the learner to discard the mentor's recommandations where those are wrong, and to trust its mentor where it is right. Figures (4.5) and (4.6) display, for

two different mentors (the same two ones as we just used), the mentor's policy and the heat-map of the confidence the learner as acquired with respect to its teacher's recommandation at the end of the learning.

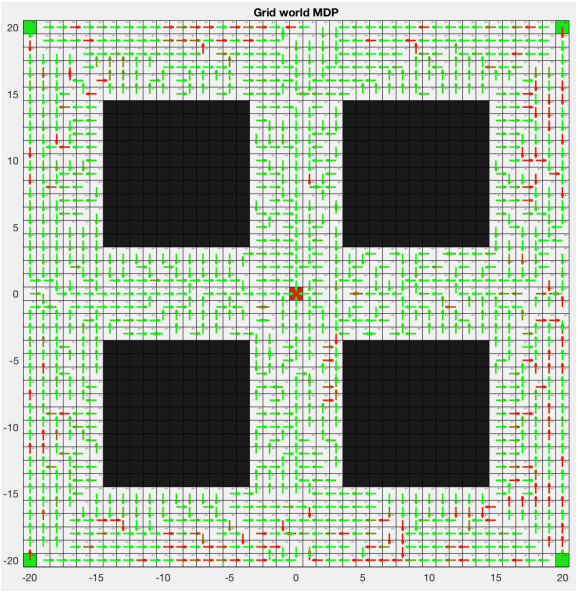


Figure 4.5: Learnt confidence : green arrows show near 1 posterior mean, red arrows near 0

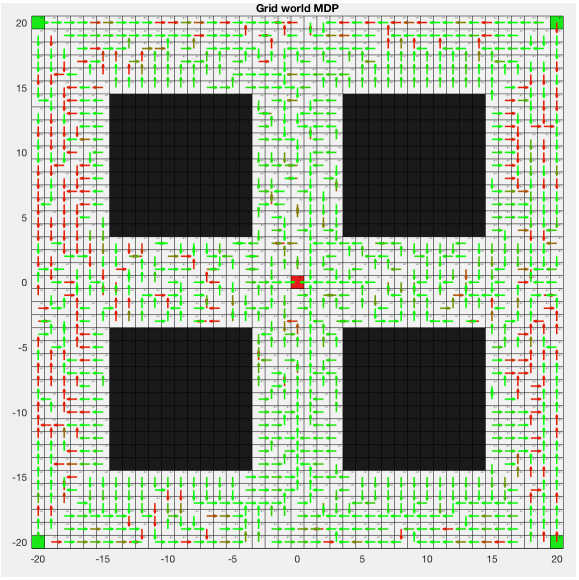


Figure 4.6: Learnt confidence : green arrows show near 1 posterior mean, red arrows near 0

The first observation one can make is that most of the suboptimal mentor actions are indeed well classified by the learner (red arrows). Also, most of the actions leading to such actions are also classified as poor recommandations, because they have a tendency of leading to deadlocks, or suboptimal actions.

4.2 Explicit compliance

4.2.1 Practical considerations

The implementation of this method is rather straight forward and contains no major difficulties. However, some hyper-parameters have to be tuned, like the two temperatures (one for the initial MDP, the other for the above action selection) as well as their dynamics, and the update rule of (3.12).

In practice, this tuning is rather easy to perform, as long as one make sure that the initial values for $Q_c(s, l)$ and $Q_c(s, d)$ $(s \in \mathcal{S})$ don't absorb the observations but also retain some prior knowledge along the learning (the learning rate β has to be related to those initial values in some way).

4.2.2 Results

Figures (4.7) and (4.8) show the learning curves for an agent using the action-value compliant-based method, with two different suboptimal teachers.

As for the β-implicit method, we are able to reduce or even suppress the undershoot that the vanishing compliance method displayed, and to obtain fast convergence.

4.2.3 Discussion

Similar plots of posterior result as for the actor-critic can now be drawn. Figures (4.9) and (4.10) display the histograms distributions for the action 'listen' and 'discard' (computed by the sigmoid value at end temperature). Figures (4.11) and (4.12) show the learnt decisions over the action-state space.

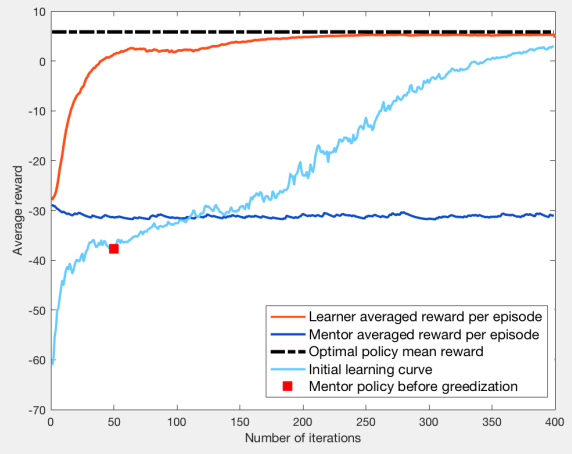


Figure 4.7: Actor-critic learnt compliance learning curve for

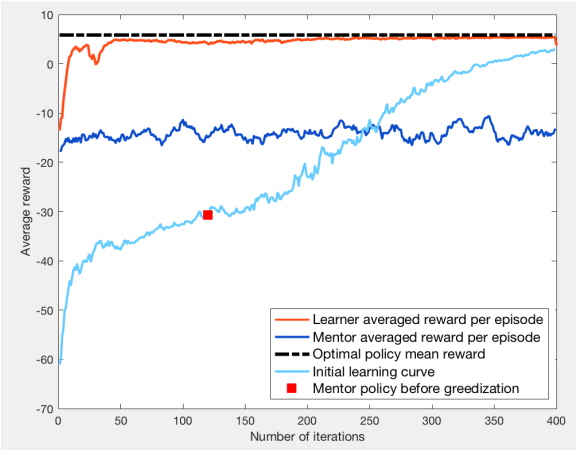


Figure 4.8: Actor-critic learnt compliance learning curve

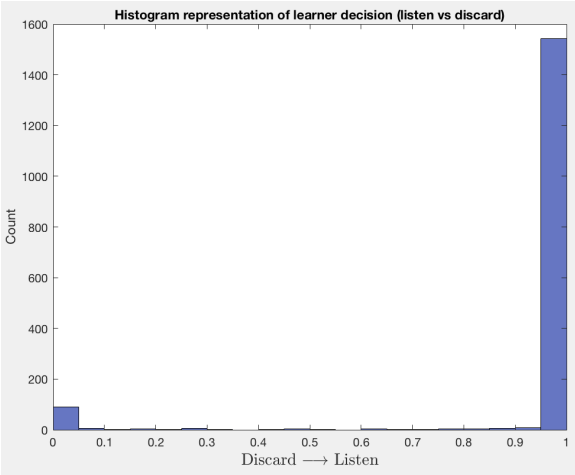


Figure 4.9: Histogram representation of the posterior decisions for a suboptimal teacher

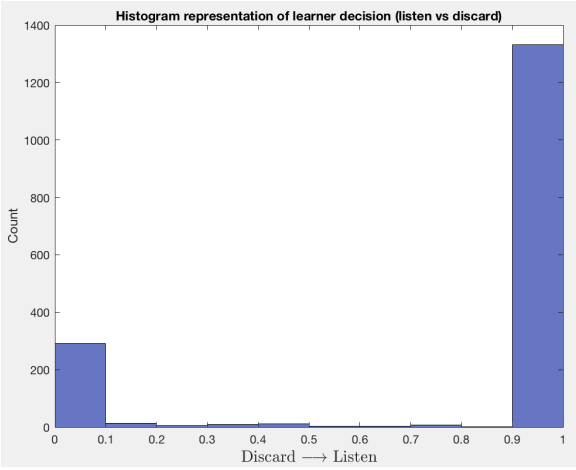


Figure 4.10: Histogram representation of the posterior decisions for a even more suboptimal teacher

Again, one can see that most wrong recommandations are well-classified. However, this method as a tendency to classify an action leading to a suboptimal sequence (in the mentor's policy) as a poor action. This as for consequence that the learner will try to always to go round the mentor's suboptimal recommandation, and sometime by making some detours.

4.3 Algorithms comparaison

We hereinafter focus on comparing the last three methods (vanishing compliance, implicit β-compliance and explicit compliance) between themselves and with classical reinforcement learning algorithms.

4.3.1 Compliant learners

Let us now compare the different compliant learners between them. Figures (4.13) and (4.14) display the learning curves derived in the previous section altogether. If the β-implicit and the explicit compliance methods seem to behave better than the vanishing learner, it seems that the speed of convergence of all three algorithms tends to equalize as the teacher sub-optimality grows. This behavior is confirmed by figures (4.15) and (4.16).

In figures (4.15) and (4.16), different teachers are being tested based on their optimality level. This scalar measure of optimality is computed from a linear scaling between the random policy and the optimal

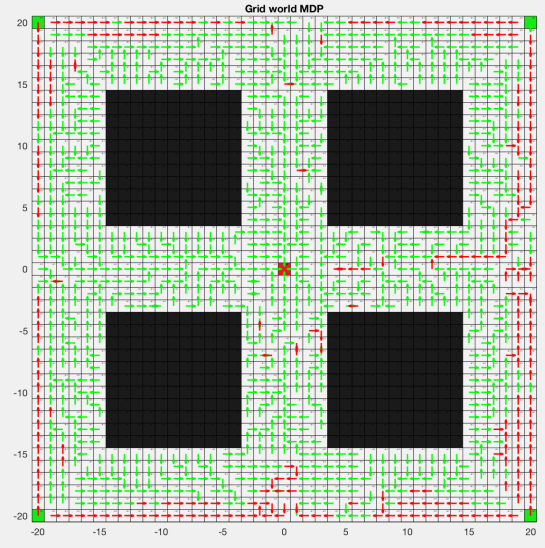


Figure 4.11: Learnt decisions : green arrows show listening, red arrows discarding

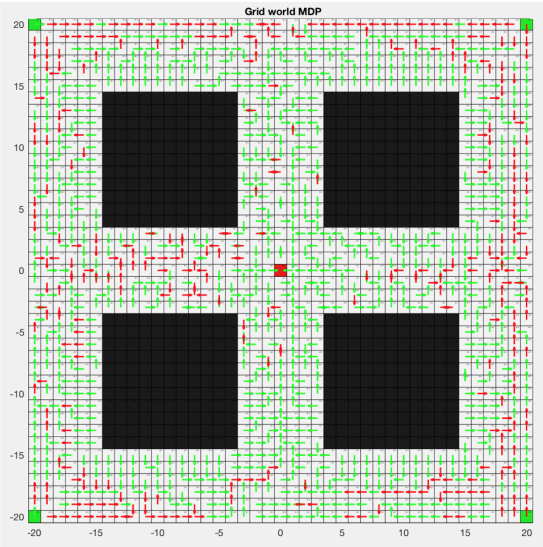


Figure 4.12: Learnt decisions : green arrows show listening, red arrows discarding

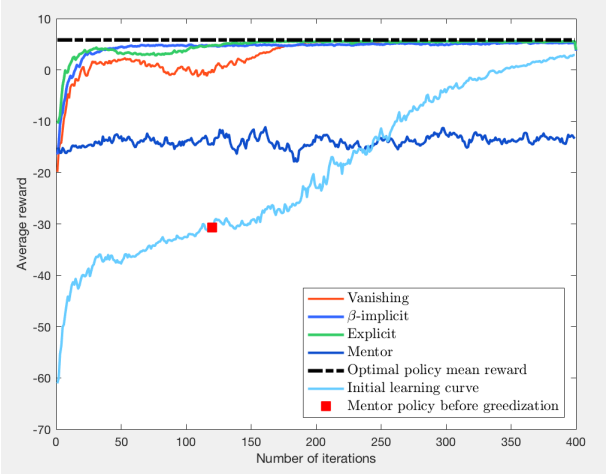


Figure 4.13: Learning curves for a first teacher

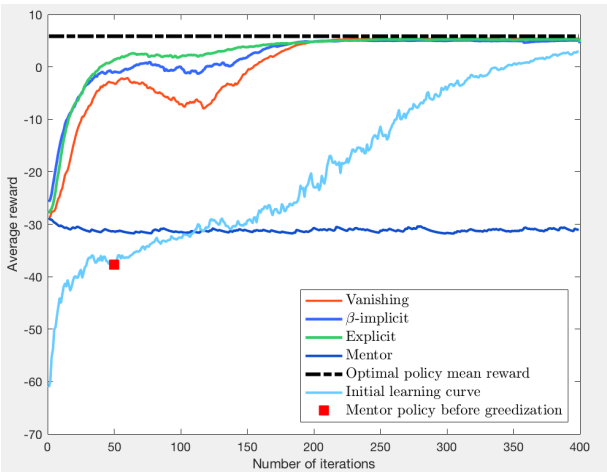


Figure 4.14: Learning curve for a second teacher

policy mean expected return on one run. As one can see, the time to convergence for all methods seem to equalize as the optimality of the teacher decreases. However, the average accumulated reward until convergence is always better for the two adaptive methods - traducing the absence of the undershoot and better behaving methods. Hence, if our algorithms can't always speed up the learning (compared to a naive compliant learner), they *provide better behaving agents.*

4.3.2 Classical learners

As reminded in the beginning of this document, one of the goals of learning to demonstration is to speed up the learning. The goal of this section is to compare the behavior of the compliant-based learners we developed with some more classical reinforcement learning.

Figures (4.17) and (4.18) display the learning curves of the compliant learner opposed with, respectively, TD(0) learners and TD(λ) learners. As one can see, our algorithms performs way better than TD(0) learners, even with largely suboptimal teachers. However, they performed as well or even slightly worse than TD(λ) learners. This phenomenon is mostly due to the fact the behind its action selection, our learners update their Q-values thanks to SARSA updates (on-policy). Generalizing to an off-policy update, and eventually by making use of eligibility traces will most likely improve the learning speed and beat TD(λ) learners.

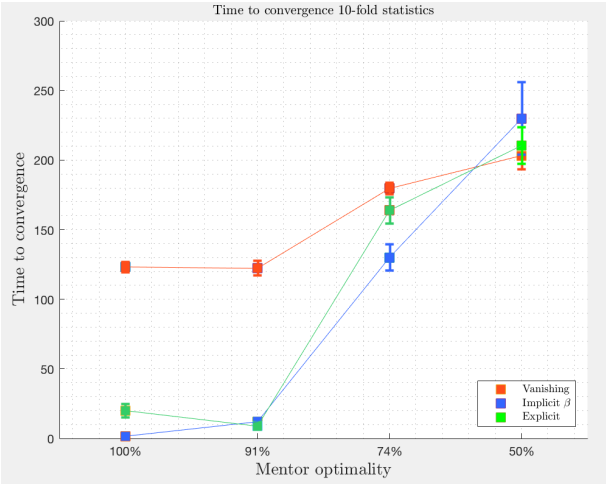


Figure 4.15: Time to convergence metric

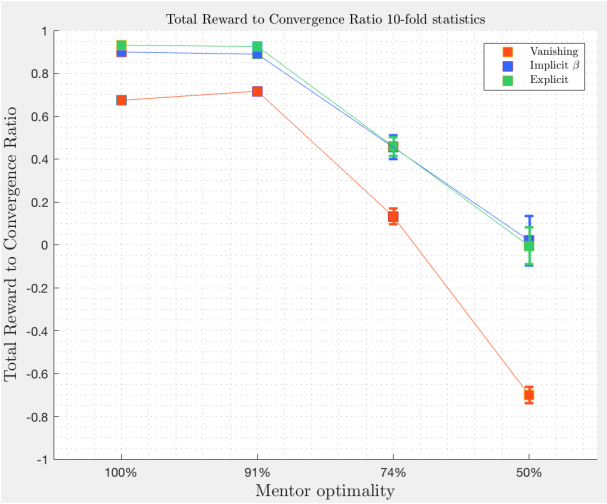


Figure 4.16: Reward to convergence metric

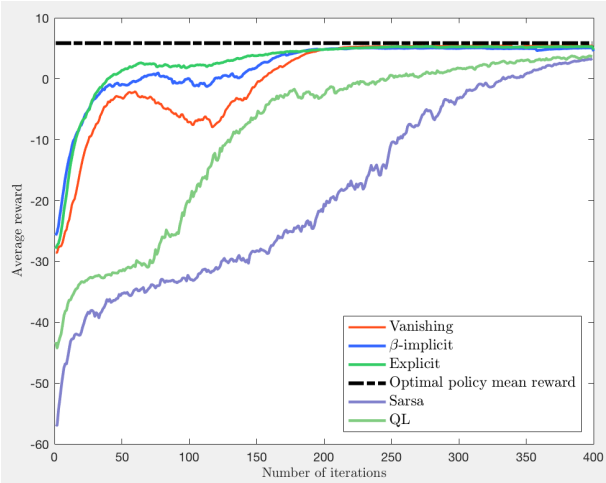


Figure 4.17: Method comparaison (teacher optimality : 50%)

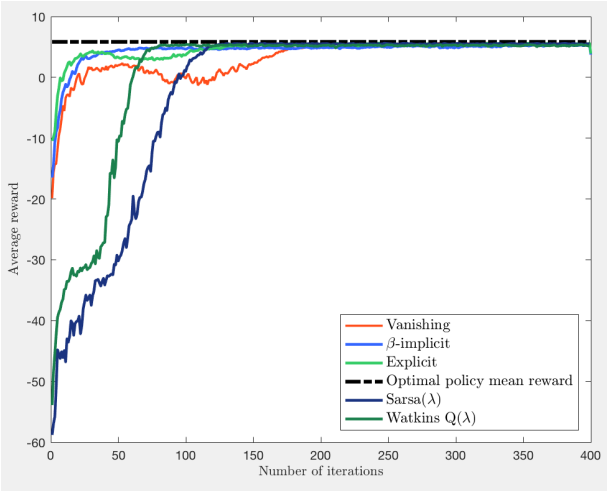


Figure 4.18: Method comparaison (teacher optimality : 75%)

4.4 Improvements

Conclusion

4.5 Outline

4.6 Applicability

4.7 Possible improvements

Bibliography

[ACVB09] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[BCD16] Aude G. Billard, Sylvain Calinon, and Rüdiger Dillmann. *Learning from Humans*, pages 1995–2014. Springer International Publishing, Cham, 2016.

[PB03] Bob Price and Craig Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[TS09] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.