Reinforcement learning with raw image pixels as input state

Damien Ernst†, Raphaël Marée and Louis Wehenkel

{ernst,maree,lwh}@montefiore.ulg.ac.be
Department of Electrical Engineering and Computer Science
Institut Montefiore - University of Liège
Sart-Tilman B28 - B4000 Liège - Belgium
† Postdoctoral Researcher FNRS

Abstract. We report in this paper some positive simulation results obtained when image pixels are directly used as input state of a reinforcement learning algorithm. The reinforcement learning algorithm chosen to carry out the simulation is a batch-mode algorithm known as fitted Q iteration.

1 Introduction

Reinforcement learning (RL) is learning what to do, how to map states to actions, from the information acquired from interaction with a system. In its classical setting, the reinforcement learning agent wants to maximize a long term reward signal and the information acquired from interaction with the system is a set of samples, where each sample is composed of four elements: a state, the action taken while being in this state, the instantaneous reward observed and the successor state.

In many real-life problems, such as robot navigation ones, the state is made of visual percept. Up to now, the standard approach for dealing with visual percept in the reinforcement learning context is to extract from the images some relevant features and use them, rather than the raw image pixels, as input state for the RL algorithm (see e.g. [3]). The main advantage of this approach is that it leads to a reduction of the input space for the RL algorithm which eases the problem of generalization to unseen situations. Its main drawback is that the feature extraction phase needs to be adapted to problem specifics.

Recently, several research papers have shown that in the image classification framework, it was possible to obtain some excellent results by applying directly state-of-the-art supervised learning algorithms (e.g. tree-based ensemble methods, SVMs) on the image pixels (see e.g. [5]). Also, recent developments in reinforcement learning have led to some new algorithms which allow to take full advantage, in the reinforcement learning context, of the generalization performances of any supervised learning algorithm [1, 4]. We may therefore wonder whether using one of these new RL algorithms directly with the raw image pixels

as input state, without any feature extraction, could lead to some good performances. In a first attempt to answer this question, we carried out simulations and report in this paper our preliminary findings.

The next section of this paper is largely borrowed from [1] and introduces, in the deterministic case, the fitted Q iteration algorithm used in our simulations. Afterwards, we describe the test problem and discuss the results obtained in various settings. And, finally, we conclude.

2 Learning from a set of samples

2.1 Problem formulation

Let us consider a system having a deterministic *discrete-time dynamics* described by:

$$x_{t+1} = f(x_t, u_t) \quad t = 0, 1, \ldots \tag{1}$$

where for all t, the state x_t is an element of the state space X, the action u_t is an element of the action space U.

To the transition from t to $t+1$ is associated an instantaneous *reward signal* $r_t = r(x_t, u_t)$ where $r(x, u)$ is the reward function bounded by some constant B_r.

Let $\mu(\cdot) : X \to U$ denote a stationary control policy and J^μ denote the return obtained over an infinite time horizon when the system is controlled using this policy (i.e. when $u_t = \mu(x_t), \forall t$). For a given initial condition $x_0 = x$, J^μ is defined as follows:

$$J^\mu(x) = \lim_{N \to \infty} \sum_{t=0}^{N-1} \gamma^t r(x_t, \mu(x_t)) \tag{2}$$

where γ is a discount factor ($0 \leq \gamma < 1$) that weighs short-term rewards more than long-term ones. Our objective is to find an optimal stationary policy μ^*, i.e. a stationary policy that maximizes J^μ for all x.

Reinforcement learning techniques do not assume that the system dynamics and the cost function are given in analytical (or even algorithmic) form. The sole information they assume available about the system dynamics and the cost function is the one that can be gathered from the observation of system trajectories. Reinforcement learning techniques compute from this an *approximation* $\hat{\pi}^*_{c,T}$ of a T-stage optimal (closed-loop) policy since, except for very special conditions, the exact optimal policy can not be decided from such a limited amount of information.

The *fitted Q iteration* algorithm on which we focus in this paper, actually relies on a slightly weaker assumption, namely that a set of one step system transitions is given, each one providing the knowledge of a new sample of information (x_t, u_t, c_t, x_{t+1}) that we name four-tuple. We denote by $\mathcal{F}$ the set $\{(x^l_t, u^l_t, c^l_t, x^l_{t+1})\}_{l=1}^{\#\mathcal{F}}$ of available four-tuples.

2.2 Dynamic programming results

The sequence of Q_N-functions defined on $X \times U$

$$Q_N(x,u) = r(x,u) + \gamma \max_{u' \in U} Q_{N-1}(f(x,u),u') \forall N > 0$$

with $Q_0(x,u) \equiv 0$ converges, in infinity norm, to the Q-function, defined as the (unique) solution of the Bellman equation:

$$Q(x,u) = r(x,u) + \gamma \max_{u' \in U} Q(f(x,u),u') \tag{3}$$

A policy μ^* that satisfies

$$\mu^*(x) = \arg\max_{u \in U} Q(x,u) \tag{4}$$

is an optimal stationary policy.

Let us denote by μ_N^* the stationary policy

$$\mu_N^*(x) = \arg\max_{u \in U} Q_N(x,u) \quad . \tag{5}$$

The following bound on the suboptimality of μ_N^* holds:

$$\|J^{\mu^*} - J^{\mu_N^*}\|_\infty \leq \frac{2\gamma^N B_r}{(1-\gamma)^2} \quad . \tag{6}$$

2.3 Fitted Q iteration

The fitted Q iteration algorithm computes from the set of four-tuples $\mathcal{F}$ the functions $\hat{Q}_1$, $\hat{Q}_2$, ..., $\hat{Q}_N$, approximations of the functions Q_1, Q_2, ..., Q_N defined by Eqn (3), by solving a sequence of standard supervised learning regression problems. The policy

$$\hat{\mu}_N^*(x) = \arg\max_{u \in U} \hat{Q}_N(x,u) \tag{7}$$

is taken as approximation of the optimal stationary policy. The training sample for the kth problem ($k \geq 1$) of the sequence is

$$((x_t^l, u_t^l), r_t^l + \gamma \max_{u \in U} \hat{Q}_{k-1}(x_{t+1}^l, u)), l = 1, \ldots, \#\mathcal{F} \tag{8}$$

with $\hat{Q}_0(x,u) = 0$ everywhere. The supervised learning regression algorithm produces from this training sample the function $\hat{Q}_k$ that is used to determine the next training sample and from there, the next function of the sequence.

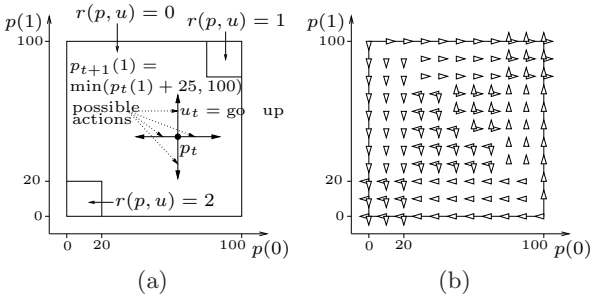


Fig. 1. Figure (a) gives information for the position dynamics and the reward function for the agent navigation problem. Figure (b) plots the optimal policy $\mu^*(p)$ for the values of $p \in \{0, 10, \ldots, 100\} \times \{0, 10, \ldots, 100\}$. Orientation of the triangle for a position p gives information about the optimal action(s) $\mu^*(p)$.

3 Simulation results

3.1 The test problem

Experiments are carried out on the navigation problem whose main characteristics are illustrated on Figure 1a. An agent navigates in a square and the reward he gets is function of his position in the square. He can at each instant t either decide to go up, down, left of right ($U = \{up, down, left, right\}$). We denote by p the position of the agent. The horizontal (vertical) position of the agent $p(0)$ ($p(1)$) can vary between 0 and 100 with a step of 1. The set of possible positions is $P = \{0, 1, 2, \ldots, 100\} \times \{0, 1, 2, \ldots, 100\}$. When the agent decides at time t to go in a specific direction, he moves 25 steps at once in this direction, unless he is stopped before by the square boundary.

The reward signal r_t observed by the agent is always zero, except if the agent is at time t in the upper right part of the square and the lower left part of the square where reward signals of 1 and 2 are observed, respectively ($B_r = 2$). The decay factor γ is equal to 0.5. The optimal policy, plotted on Figure 1b drives the agent to one of these corners. Even if larger reward signals are observed the lower left corner, the optimal policy does not necessarily drive the system to this corner. Indeed, due to the discount factor γ that weighs short-term reward signal more than long-term ones, it may be preferable to observe smaller reward signals but sooner.

Our goal is to study the performances of the fitted Q iteration algorithm when the input state for the RL algorithm is not the position p but well a visual percept. In this context, we represent on top of the navigation square a *navigation image*, and we have supposed that when being at position p, the agent has access to the visual percept $pixels(p)$ which is a vector of pixel values that encodes the image region surrounding its position p (Figure 2). The matrix giving the grey levels of the 100 tiles of Figure 2 is given in 5.

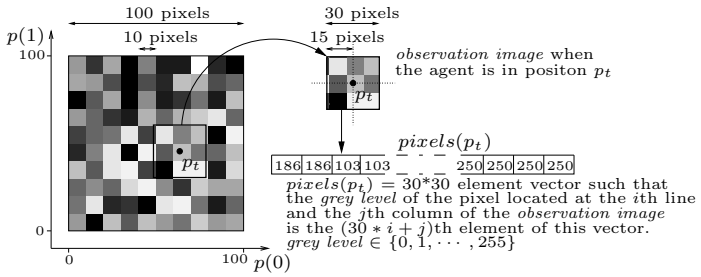


Fig. 2. Visual percept $pixels(p_t)$ for the agent when being position p_t. Pixels of the 30×30 *observation image* which are not contained in the 100×100 *navigation image*, which happens when $p_t(i) < 15$ and/or $p_t(i) > 85$, are assumed to be black pixels (*grey level*=0).

In our study, we have partionned the 100×100 navigation image into one hundred 10×10 subimages that we name *tiles*. For every tile, we have selected a grey level at random in $\{0, 1, \ldots, 255\}$ and set all its pixels to this grey level. After having generated the image, we have checked whether two different positions p were indeed leading to different vectors of pixel values $pixels(p)$. This check has been done in order to make sure that considering $pixels(p)$ rather than p as input state does not lead to a partially observable system.

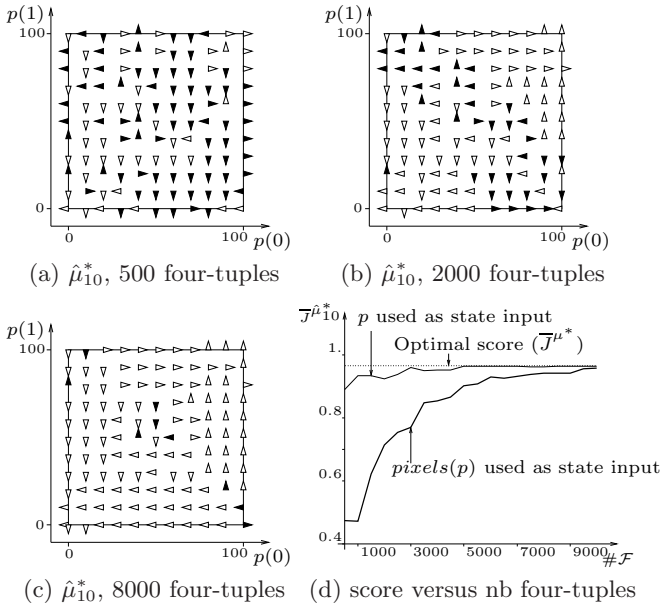
3.2 Four-tuples generation

To generate the four-tuples we consider one step episodes with the initial position for each episode being chosen at random among the $101 * 101$ possible positions p and the action being chosen at random among U. More precisely, to generate a set $\mathcal{F}$ with n elements, we repeat n times the sequence of instructions:
1. draw p_0 at random in P and u at random in U;
2. observe r_0 and p_1;
3. add $(pixels(p_0), u_0, r_0, pixels(p_1))$ to $\mathcal{F}$.

3.3 Fitted Q iteration algorithm

Within the fitted Q iteration algorithm, we have used in our simulations a regression tree based ensemble method called Extra-Trees [2].[1] To apply this algorithm at each iteration, the training sample defined by Eqn (8) is split into four subsamples according to the four possible values of u, and $\hat{Q}_k(x, u)$ for each value of u is obtained by calling the Extra-Trees algorithm on the corresponding subsample.

[1] The Extra-Trees algorithm has three parameters M (the number of trees that are built to define the ensemble model), $n_{\min}$ (the minimum number of samples of non-terminal nodes) and K (the number of cut-directions explored to split a node). They have been set to $M = 50$, $n_{\min} = 2$ (yielding fully developed trees) and $K = 900$ (equal to the dimensionality of the input space).



(a) $\hat{\mu}^*_{10}$, 500 four-tuples (b) $\hat{\mu}^*_{10}$, 2000 four-tuples

(c) $\hat{\mu}^*_{10}$, 8000 four-tuples (d) score versus nb four-tuples

Fig. 3. Figures (a-c) plot the policy $\hat{\mu}^*_{10}$ computed for increasing values of $\#\mathcal{F}$. The orientation of the triangles indicate the value of $\hat{\mu}^*_{10}(pixels(p))$; white triangles indicate that it coincides with an optimal action. Figure (d) plots the score of the policies: the horizontal line indicates the score of the optimal policy, the darker curve (with smaller scores) corresponds to the case of pixel based learning with growing number of four-tuples, while the lighter curve provides the scores obtained with the same samples when the position p is used as state representation.

The number of iterations N of the fitted Q iteration algorithm is chosen equal to 10, leading to an upper bound of 0.015625 in Eqn (6) which is tight enough for our purpose, since $J^{\mu^*}(x) \in [0.25, 4]$. The policy $\hat{\mu}^*_{10}(x) = \arg\max_u \hat{Q}_{10}(x, u)$ is taken as approximation of the optimal stationary policy.

3.4 Results

Figures 3a-c show how the policies $\hat{\mu}^*_{10}$ change when increasing the size of the set of four-tuples. In particular, Figure 3c shows that with 8000 four-tuples, the policy almost completely coincides with the optimal policy μ^* of Figure 1b. To further assess the speed of convergence of the algorithm, we have plotted on Figure 3d the score[2] of policies obtained in different conditions. We observe that with respect to the compact state representation in terms of positions, the use

[2] The score of a policy is defined here as the average value over all possible initial states of the return obtained over an infinite time horizon when the system follows this policy.

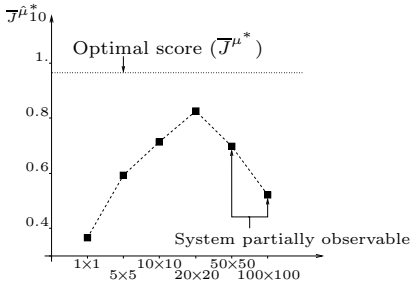


Fig. 4. Evolution of the score with the size of the constant grey level tiles. $\#\mathcal{F} = 2000$.

of the pixel based representation slows down but does not prevent convergence. Indeed, with $\#\mathcal{F} = 10,000$ the score of the pixel based policy has almost converged to the optimal one, which is a fairly small sample size if we compare it to the dimensionality of the input space of 900. This suggests that the fitted Q iteration algorithm coupled with Extra-Trees is able to cope with a low-level representation where information is scattered in a rather complex way over a large number of input variables.

To illustrate the influence of the navigation image characteristics on the results, we carried out an experiment where we have changed the size of the constant grey level tiles while keeping constant the size of the observation images. The results, depicted on Figure 4, show that the score first increases, reaches a maximum, and decreases afterwards.

To explain these results, we first notice that the Extra-Trees method works by inferring from a sample $\mathcal{TS} = ((i^l, o^l), l = 1, \ldots, \#\mathcal{TS})$ a kernel $K(i, i')$, from which an approximation of the output o associated with an input i is computed by $\hat{o}(i) = \sum_{(i^l, o^l)} K(i, i^l) o^l$. The value of $K(i, i^l)$ thus determines the importance of the output o^l in the prediction, and for our concern the main property of the Extra-Trees kernel is that it takes larger values if the vectors i and i^l have many components which are close to each other, i.e. if there exists many values of $j \in \{1, 2, \cdots, \text{size of vector } i\}$ such that $i[j]$ is close from $i^l[j]$ [2]. Next, we note from Figure 3d that when the algorithm is applied to a training set of size 2000 with positions as inputs (i.e. $\mathcal{TS}_p = ((p^l, o^l), l = 1, \ldots, \#\mathcal{TS}_p)$) it provides close to optimal scores. With this input representation, elements (p^l, o^l) such that p^l is geometrically close to p tend to lead to a high value of $K(p, p^l)$ and one may therefore reasonably suppose that when using pixel vectors as inputs, good results will be obtained only if the resulting kernel $K(pixels(p), pixels(p^l))$ is strongly enough correlated with the geometrical distance between p and p^l, which means that the closer two positions the more similar the corresponding vectors of pixel values should be.

With this we can explain the influence of the size of the tiles on the score in the following way. Let p^l be a position such that its 30×30 observation image

is fully contained in the square. Then, when the navigation image is composed of randomly chosen 1×1 tiles, for a position $p \neq p^l$ there is no reason that the value of $K(pixels(p), pixels(p^l))$ should depend on the geometrical distance between p and p^l. In other words, the kernel derived in these conditions will take essentially only two values, namely $K(pixels(p), pixels(p^l)) = 1$ if $p = p^l$ and $K(pixels(p), pixels(p^l)) \approx 1/\#\mathcal{TS}$ otherwise. Thus, the output predicted at a position far enough from the square boundary will essentially be the average output of the training set, except for positions contained in the training sample. When the tiles become larger, the dependence of the amount of similar pixels of two observation images on their geometrical distance increases, which leads to a more appropriate approximation architecture and better policies. However, when the tiles size becomes too large the sensitivity of the pixel based kernel with respect to the geometrical distance eventually decreases. In particular, the loss of observability above a certain tiles size translates into a dead-band within which the kernel remains constant, which implies suboptimality of the inferred policy, even in asymptotic conditions.

4 Conclusions

We have applied in this paper a reinforcement learning algorithm known as fitted Q iteration to a problem of navigation from visual percepts. The algorithm uses directly as state input the raw pixel values. The simulation results show that in spite of the fact that in these conditions the information is spread in a rather complex way over a large number of low-level input variables, the reinforcement learning algorithm was nevertheless able to converge relatively fast to near optimal navigation policies. We have also highlighted the strong dependence of the learning quality on the characteristics of the images the agent gets as input states, and in particular on the relation between distances in the high-dimensional pixel-based representation space and geometrical distances related to the physics of the navigation problem.

5 Image description

We provide hereafter the 10×10 matrix giving the grey levels of the 100 tiles of Figure 2:

$$\begin{bmatrix}
164 & 55 & 175 & 6 & 132 & 27 & 35 & 255 & 47 & 11 \\
169 & 155 & 87 & 5 & 77 & 39 & 197 & 179 & 82 & 111 \\
5 & 92 & 176 & 10 & 148 & 37 & 57 & 119 & 32 & 193 \\
156 & 110 & 54 & 38 & 186 & 103 & 190 & 212 & 241 & 108 \\
65 & 103 & 125 & 239 & 73 & 235 & 128 & 199 & 3 & 247 \\
42 & 129 & 233 & 3 & 250 & 101 & 196 & 119 & 108 & 192 \\
199 & 91 & 240 & 254 & 71 & 2 & 250 & 250 & 36 & 227 \\
109 & 150 & 111 & 224 & 244 & 152 & 57 & 205 & 173 & 174 \\
124 & 242 & 42 & 62 & 0 & 234 & 252 & 127 & 28 & 114 \\
163 & 7 & 198 & 92 & 192 & 163 & 115 & 208 & 160 & 168
\end{bmatrix}$$

References

1. D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, April 2005.
2. P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 36(1):3–42, 2006.
3. S. Jodogne and J. Piater. Interactive learning of mappings from visual percepts to actions. In L. De Raedt and S. Wrobel, editors, *Proceedings of the 22nd International Conference on Machine Learning*, pages 393–400, August 2005.
4. M. Lagoudakis and R. Parr. Reinforcement learning as classification: leveraging modern classifiers. In T. Faucett and N. Mishra, editors, *Proceedings of 20th International Conference on Machine Learning*, pages 424–431, 2003.
5. R. Marée, P. Geurts, J. Piater, and L. Wehenkel. Random subwindows for robust image classification. In C. Schmid, S. Soatto, and C. Tomasi, editors, *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, volume 1, pages 34–40. IEEE, June 2005.