REINFORCEMENT LEARNING OF PERCEPTUAL CLASSES USING Q LEARNING UPDATES

Sébastien Jodogne*
Montefiore Institute (B28)
University of Liège
B-4000 Liège, Belgium
email: S.Jodogne@ULg.ac.be

Justus H. Piater
Montefiore Institute (B28)
University of Liège
B-4000 Liège, Belgium
email: Justus.Piater@ULg.ac.be

ABSTRACT
We introduce a new Reinforcement Learning algorithm designed to operate in perceptual spaces upon which it is possible to define features, such as visual spaces. It incrementally constructs a classifier that maps the percepts to a perceptual class by testing the presence or absence of highly informative features. This approach has the advantage of enhancing the generalization abilities of the autonomous agent, while reducing the influence of noise, as well as the size of the perceptual domain.

The perceptual classes are iteratively refined using a statistical analysis of the updates that Q Learning would apply to an optimal Q function for the considered classification. This process relies only on the reinforcements earned by the agent during its interaction with the environment. Thus, the distinctive features are selected interactively in a task-driven fashion, without an external supervisor.

KEY WORDS
Machine Learning, Computer Vision, Feature Selection.

1 Introduction

One of the most exciting topics in robotics is to design autonomous agents that automatically learn to behave properly in some *a priori* unknown environment by extrapolating from their interactions with this environment. This challenging goal gave rise to the theory of *Reinforcement Learning* (RL) that allowed to better understand the mechanisms by which living beings learn to solve their everyday tasks [2]. In RL, each time the agent takes a decision, it receives a reward or a punishment, called the *reinforcement*, that evaluates its reaction. RL has had spectacular applications in robotic control, e.g. making a quadruped robot learn walking progressively without any human intervention [6].

Studies in RL have shown deficiencies when the input and/or output spaces of the agent is large or noisy. Since the rise of embedded CCD sensors, this problem has been exacerbated. Things get even worse if those spaces are continuous, which is the case for example if the agent controls robotic actuators with multiple degrees of freedom. In such continuous cases, the traditional solution is to discretize the spaces. However, this results in an explosion of the representational size of the domains known as the *curse of dimensionality*. Recent research seeks to take advantage of powerful supervised-learning methods that handle high-dimensional problems with excellent generalization performances [4, 11].

With the same motivation in mind, we have introduced the paradigm of *Reinforcement Learning of Classes* (RLC) [7]. RLC uses context-dependent information to reduce the size of the input space[1] by focusing the attention of the agent on pertinent and robust parts of the raw percepts. This enhances the rate of convergence as well as the robustness to noise of RL algorithms. Such highly informative patterns in the inputs are called *features*, and are drawn from a possibly infinite set. The features we consider are *binary*, in the sense that a given percept can exhibit them or not. For example, in visual perceptual spaces, features could be local descriptions of patches of images [9]. In this case, an image is considered to exhibit a visual feature if there exists an interest point in the image such that the description of its neighborhood is similar to the feature.

RLC is founded on a two-level architecture: (i) A classifier partitions the perceptual space into *perceptual classes* by testing the presence of suitably selected features, then (ii) an embedded RL algorithm is fed with the output of the classifier. If the agent identifies inconsistencies in the earned reinforcements when faced with some perceptual class (e.g., if the same reaction leads sometimes to a reward, and other times to a punishment), it informs the classifier that it should refine itself by selecting a new distinctive feature in the percepts corresponding to that class.

The agent will consequently learn perceptual classes only through interactions in a task-driven fashion, which is the central property of this system. Contrary to classical techniques for discretizing perceptual spaces [16], we do not assume that the interesting features, nor their number, are fixed in advance. Rather, our algorithms dynamically learn new features when needed. This also results in a good interpretability of the results.

The main contribution of this paper is the introduction of a new RLC algorithm, called *RLC through Q Learning Updates*, that does not make any assumption about the way

*Research fellow of the Belgian National Fund for Scientific Research.

[1] In the sequel, perceptual space is a synonym for input space.

experiments are acquired, contrary to the algorithm *RLC through Sequences of Actions*, that was described in our previous work [7]. This is a great advantage, especially in robotic applications, in which collecting a representative set of experiments can be very expensive. Moreover, a stopping criterion can be derived for the new algorithm, which was not possible previously. We finally present the results of the new algorithm on a visual navigation task.

2 Theoretical Background

2.1 Markov Decision Problems

In RL, the environment is generally modeled as a *Markov Decision Process* (MDP). A MDP is a tuple $\langle S, A, \mathcal{T}, \mathcal{R} \rangle$, where S is a finite set of states, A is a finite set of actions, $\mathcal{T}$ is a probabilistic mapping from $S \times A$ to S, and $\mathcal{R}$ is a mapping from $S \times A$ to $\mathbb{R}$. MDP obey the following discrete-time dynamics: If at time t, the agent takes the action a_t while the environment lies in a state s_t, the agent perceives a numerical reinforcement $r_{t+1} = \mathcal{R}(s_t, a_t)$, then reaches some state s_{t+1} with probability $\mathcal{T}(s_t, a_t, s_{t+1})$. Throughout this work, we will assume that the perceptual space is *fully observable*, i.e. the agent is able to distinguish between the states of the environment using only its sensors. This allows to talk equally about states and percepts.

A *stationary Markovian control policy* (for shortness, a *policy*) is a probabilistic mapping π from the states to the actions. A policy tells the agent the probability with which it should choose an action in each state.

The *Markov Decision Problem* for one given MDP is to find an *optimal policy* that maximizes an evaluation of the performances of the agent over time, called the *objective function*. The *discounted return*-based objective function is commonly used in DP. Given an infinite sequence of interactions, the discounted return at time t is:

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

where $\gamma \in [0, 1[$ is the *discount factor* that gives the current value of the future reinforcements. Therefore, the agent should not only maximize its immediate reinforcements (i.e., the sequence of r_t), but should also take less attractive actions that move it into parts of the state space where it expects obtaining higher future reinforcements. This trade-off is known as the *delayed-reward problem*.

2.2 Dynamic Programming

The theory of *Dynamic Programming* (DP) is concerned with solving Markov Decision Problems. Let us call the Q *function* of a policy π, the function giving for each state $s \in S$ and each action $a \in A$, the expected discounted return obtained by starting from the state s, taking the action a, and thereafter following the policy π:

$$Q^\pi(s, a) = \mathrm{E}_\pi \left\{ R_t \mid s_t = s,\ a_t = a \right\}, \tag{2}$$

where E_π denotes the expected value when the agent follows the policy π. Let us also define the H *mapping* from Q functions to Q functions as:

$$(HQ)(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{T}(s, a, s') \max_{a' \in A} Q(s', a'), \tag{3}$$

for all $s \in S$ and $a \in A$. Bellman [1] proved that all the optimal policies for a given MDP share the same Q function, denoted Q^*, that always exists and that satisfies Bellman's so-called optimality equation:

$$HQ^* = Q^*. \tag{4}$$

Once Q^* is known, an optimal deterministic policy π^* is easily derived by letting $\pi^*(s) = \mathrm{argmax}_{a \in A} Q^*(s, a)$ for each $s \in S$. Hence, DP reduces the Markov Decision Problem to the resolution of the non-linear system of Equations (4). However, the direct resolution is seldom used in practice, to the advantage of incremental algorithms such as *State-Action Value Iteration* [2].

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a set of algorithmic methods for solving Markov Decision Problems when the underlying MDP is unknown. The input given to RL algorithms is basically a sequence of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ of the agent with its environment. RL techniques can be roughly divided in two categories: (i) *Model-based methods*, that first build an estimation of the underlying MDP (e.g., by computing the relative frequencies that appear in the sequence of interactions), and then use classical DP algorithms, and (ii) *model-free methods*, such as *SARSA* and $TD(\lambda)$ [2], that do not compute such an estimation.

2.4 Q Learning

The most popular model-free method, Q *Learning* [17], incrementally improves an estimation $\hat{Q}$ of the Q^* function by applying the following update rule whenever an experiment $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ is acquired:

$$\hat{Q}(s_t, a_t) := \hat{Q}(s_t, a_t) + \alpha_t \Delta_t, \tag{5}$$

where $\alpha_t \in [0, 1]$ is known as the *learning rate* at time t, and Δ_t is the Q *Learning update* to be applied to $\hat{Q}(s_t, a_t)$:

$$\Delta_t = r_{t+1} + \gamma \max_{a' \in A} \hat{Q}(s_{t+1}, a') - \hat{Q}(s_t, a_t). \tag{6}$$

Under some mild assumptions about the learning rate and the way experiments are acquired, it was proved that random fluctuations in $\hat{Q}$ due to $\mathcal{T}$ are progressively eliminated as the number of experiments tends to infinity.

There is a strong relation between Δ_t and the H mapping. If the transition function $\mathcal{T}$ is deterministic, the Q Learning update can be rewritten as $\Delta_t = (H\hat{Q})(s_t, a_t) - \hat{Q}(s_t, a_t)$. Hence, by Bellman's optimality equation, from the moment where $\hat{Q}$ becomes equal to Q^*, the Q Learning updates will always be zero.

3 Reinforcement Learning of Classes

As discussed in the Introduction, *Reinforcement Learning of Classes* (RLC) aims at building a classifier that maps percepts to symbolic perceptual classes by testing the presence of context-dependent features. The output of the classifier is connected to the input of an embedded RL algorithm that controls the agent. Such an approach requires the presence of at least two elements in the perceptual space: (i) A possibly infinite set of *(binary) features*, which are properties that can be exhibited or not by the raw percepts, and (ii) a *feature detector*, which is a Boolean function that tells whether or not a given feature is present in a given percept.

At any time k, the classifier, which will be denoted $\mathcal{C}_k$, partitions the perceptual space into a number m_k of *perceptual classes* $\{P_1, \ldots, P_{m_k}\}$ using a finite number of selected, highly informative features. Each class P_i is identified by a Boolean conjunction of features. All these Boolean formulas must be mutually exclusive. Such a classifier can be implemented as a binary decision tree, the leaves of which correspond to the perceptual classes, and the internal nodes of which test the presence of one feature.

Initially, the classifier just knows about one class. Of course, this introduces a kind of *perceptual aliasing*: Percepts requiring different reactions may be merged into the same class. Perceptual aliasing reveals itself in inconsistencies between discounted returns in a given class. Techniques for detecting and removing (or avoiding) perceptual aliasing have been studied in the literature [3, 8, 18]. However, such techniques are irrelevant in our context, in which aliasing can be removed by further refining the classifier. In fact, previous approaches address a *lack of information* inherent to the used sensors, whereas ours handles a *surplus of information* related to redundancies in the input space.

The challenge in RLC is therefore to identify the aliased classes, then to refine them by selecting new distinctive features. This is the topic of the following sections. The refining process should use no more information than available in the context of RL, i.e. a sequence of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$. Eventually, when no further refinement is needed, the classifier is said *complete*. Here, we implicitly assume that percepts requiring different reactions can always be distinguished using a finite Boolean conjunction of features. Intuitively, this simply means that the used features should be powerful enough for the task.

3.1 Identifying Aliased Classes

Let P_i a perceptual class. We propose two ways for detecting aliasing in P_i, that are not described in the seminal papers about perceptual aliasing [3, 8, 18]. For the time being, the transition function $\mathcal{T}$ is assumed deterministic.

3.1.1 Sequences of Actions

A possible solution is to check if there exists a finite sequence of actions that gives rise to two different sequences of reinforcements for two states in P_i. This basic idea has been exploited in the algorithm *RLC through Sequences of Actions* [7]. Since the criterion can only use the interactions that the agent has previously encountered, it is in practice impossible to consider every possible sequence of actions.

We solved this problem by requiring the agent to choose one sequence of actions for each perceptual class. Every time a perceptual class is met, either its associated sequence of actions is executed, or a random action is taken to explore new parts of the input space. This ensures that, in the absence of perceptual aliasing, the same sequence of actions will be generated whenever the agent decides to perform its deterministic behavior when faced with a perceptual class P_i. Periodically, the agent selects new sequences of actions. This can be done either by randomization, or by using interesting trajectories in the system according to the information gathered on the task so far.

The main problem with this approach is that it imposes strong requirements on the way the interactions are acquired. For instance, it prevents the use of a static database of interactions. This is particularly restrictive in robotic applications, where collecting a representative set of experiments can be very expensive. Furthermore, it is impossible to know when the classes have been sufficiently refined. Finally, the power of this aliasing criterion strongly depends on the way the sequences of actions are chosen. This motivates the introduction in this paper of a more flexible measure of the inconsistencies in a partition P_i.

3.1.2 Q Learning Updates

The classifier $\mathcal{C}_k$ converts an input sequence of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ into a *mapped sequence* of tuples $\langle \mathcal{C}_k(s_t), a_t, r_{t+1}, \mathcal{C}_k(s_{t+1}) \rangle$. In general, there may exist no MDP that can generate such a mapped sequence, since it does not necessarily verify the Markovian assumption anymore. So, if some RL algorithm is run on the mapped sequence, it might not converge to a useful solution with respect to the underlying spaces.

Let use define the *mapped MDP* $\mathcal{M}_k$ as the MDP $\langle S', A, \mathcal{T}', \mathcal{R}' \rangle$ obtained from the mapped sequence, where $S' = \{P_1, \ldots, P_{m_k}\}$ is the set of perceptual classes that are known to $\mathcal{C}_k$, and where $\mathcal{T}'$ and $\mathcal{R}'$ have been computed using the relative frequencies occurring in the mapped sequence. When applied on a mapped sequence, Q Learning and $TD(0)$ have been shown to converge toward the optimal Q function of the corresponding mapped MDP [14]. Of course, model-based RL methods would give the same results if they use the mapped MDP as a model.

Let us call Q'^*_k, the optimal Q function obtained by the RL process for the mapped MDP $\mathcal{M}_k$. Q'^*_k is defined on the domain $\{P_1, \ldots, P_{m_k}\} \times A$. This Q function induces another Q function on the initial domain $S \times A$, denoted Q^*_k, that is defined by the relation $Q^*_k(s, a) = Q'^*_k(\mathcal{C}_k(s), a)$ for all $s \in S$ and $a \in A$.

If the classifier $\mathcal{C}_k$ were complete, then the task could be solved using this classifier, and Q^*_k would correspond to

Q^*, according to Bellman's theorem that states the uniqueness of the optimal Q function. In this case, as $\mathcal{T}$ is assumed deterministic, the updates that Q Learning would apply to Q_k^* should be zero (cf. Section 2.4). More precisely, let P_i be a perceptual class and a an action. Then, for all time stamps t such that $\mathcal{C}_k(s_t) = P_i$ and $a_t = a$, the following quantity should be zero:

$$\Delta_t = r_{t+1} + \gamma \max_{a' \in A} Q_k^*(s_{t+1}, a') - Q_k^*(s_t, a_t). \quad (7)$$

This suggests using the Q Learning updates as a measure of the aliasing in a perceptual class. The new aliasing criterion therefore consists in computing the Q_k^* function, then to sweep again all the tuples $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ that were used to generate Q_k^*, computing the quantity Δ_t for each of them. The class P_i is considered aliased if for some action a, there exist variations in Δ_t for the t such that $\mathcal{C}_k(s_t) = P_i$ and $a_t = a$.

This criterion is different from the one described in Section 3.1.1, of which it overcomes the main problems: It provides an inherent stopping criterion, and it does not impose any requirement on the way interactions are collected.

3.2 Refining an Aliased Class

Once aliasing has been detected in a class P_i with respect to an action a, we need to discover a new feature that best explains the variations in the set of Q Learning updates Δ_t corresponding to P_i and a. This is a classification problem, for which we propose an adaptation of the standard splitting rule used when building decision trees [10].

Firstly, we sort the updates Δ_t. Each cutpoint in the sorted sequence induces a binary partition of the percepts mapped to the perceptual class P_i: those percepts s_t such that the corresponding updates Δ_t are above the cutpoint, and the others. Then, for each cutpoint, we select the feature that maximizes an information-theoretic score for this partition of percepts. Finally, only the feature that has the maximal score among all the extracted features is kept.

RLC therefore assumes that a third element can be defined on the perceptual space, namely a *refining oracle*. Given two sets of percepts, a refining oracle returns the most informative feature explaining this partition into two sets. It is introduced as an oracle since it allows to use context-dependent information to direct the search for the best feature: The oracle is not obliged to exhaustively consider every feature, which is particularly useful if there is an infinite number of features.

To refine the class P_i in the classifier $\mathcal{C}_k$ using a feature F, we simply replace P_i by two classes P_i' and P_i'', respectively labeled by the formulas $\psi \wedge F$ and $\psi \wedge \neg F$, where ψ is the Boolean conjunction corresponding to P_i. This construction is easy to achieve on a binary decision tree. It consists in replacing the leaf corresponding to P_i by an internal node testing the presence of F and leading to two new leaves corresponding to P_i' and P_i''.

3.3 Non-deterministic Environments

We have assumed since Section 3.1 that the environment behaves deterministically. Of course, this might not be the case. So, a hypothesis test using the χ^2-statistic is applied after each class refining attempt in order to decide if the selected feature induces a genuine split that is significantly different from a random split. This approach is inspired from decision tree pruning [13].

3.4 Algorithm

Putting it all together, we obtain the algorithm called *RLC through Q Learning updates*:

1. Begin with a counter $k = 1$ and a classifier $\mathcal{C}_k$ that maps all the percepts in one single class.
2. Construct an optimal Q function Q_k^* for the classification induced by $\mathcal{C}_k$:
   - either by applying Q Learning or $TD(0)$, and recording all the interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ encountered during the RL process in a database;
   - or by collecting a representative database of interactions, for example by reading a static database or by using a $\varepsilon-$greedy exploration policy based on Q_{k-1}^*, and then applying any model-based RL technique on the mapped MDP for the sequence $\langle \mathcal{C}_k(s_t), a_t, r_{t+1}, \mathcal{C}_k(s_{t+1}) \rangle$.
3. Let $\mathcal{C}_{k+1} := \mathcal{C}_k$. For each $(i, a) \in \{1, \ldots, m_k\} \times A$:
   (a) Select all the interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ in the database such that $\mathcal{C}_k(s_t) = P_i$ and $a_t = a$.
   (b) For each one of them, compute the value:
   $$\xi_t = r_{t+1} + \gamma \max_{a' \in A} Q_k^*(s_{t+1}, a').$$
   Note that since $Q_k^*(s_t, a_t)$ is a constant c for all the selected interactions, ξ_t is just the Q Learning update Δ_t from Equation (7), offset by c.
   (c) If some of the ξ_t are different from c, select a distinctive feature that best explains the variations in the set of ξ_t. If the hypothesis test succeeds, refine the classifier $\mathcal{C}_{k+1}$ by splitting the class P_i with respect to the newly selected feature.
4. If $\mathcal{C}_k \neq \mathcal{C}_{k+1}$, let $k := k + 1$ and go to step 2.

When the algorithm stops, the classifier $\mathcal{C}_k$ is assumed complete and can be used to control the system.

4 Experiments with Visual Sensors

The behavior of RLC through Q Learning updates has been investigated in the context of a simple navigation problem that makes use of visual sensors, namely escaping from a discrete 2D maze constituted of walls and empty cells. The goal of the agent is to reach as fast as possible the exit of

the maze. In each cell, the agent has four possible actions: Go up, right, down, or left. If a move would take the agent into a wall, its location is not changed. If a move takes it into the exit, the agent is randomly teleported elsewhere in the maze. The agent earns a reward of 100 when the exit is reached, and a penalty of -1 for any other move. Note that the agent is faced with the delayed-reward problem.

This task is directly inspired by Sutton's so-called "Gridworld" [15], with the major exception that our agent does not have a direct access to its (x, y) position in the maze. Rather, the position is implicitly encoded in the percepts by an image: In each cell, a different object is buried under a transparent glass, and the sensors of the agent return a color picture of the object underneath.

We have used the pictures from the COIL-100 database[2]. In our setup, features are defined as 8-dimensional vectors of reals corresponding to color differential invariants [5]. The feature detector locates all the Harris color points of interest in the input image. For each of them, it tests whether the local description of its neighborhood is similar to the input feature, with respect to Mahalanobis' metric. The refining oracle also iterates over all the Harris color points of interest found in its input pair of buckets of visual percepts. For each of those points, it evaluates the split induced by the local description of the neighborhood around the point. Finally, it returns the descriptor that induces the best split score. We have made experiments for this task under different configurations:

Small Gridworld: The topology for this first experiment is depicted in Figure 1. Figure 2 shows the obtained results. It can easily be seen that the policy built using the last classifier is indeed optimal for the task, since the algorithm succeeds at distinguishing between all the 7 visual inputs. There were 120 distinct visual features, but RLC has only selected 6 features. The algorithm stopped once k reached the value 6.

Large Gridworld: In a second experiment, we have used Sutton's original Gridworld topology [15], which is depicted in Figure 3. Here also, RLC managed in 45 steps to build a classifier that allows the agent to distinguish between all the states, and therefore to optimally solve its task. This classifier is too large to be included in this paper. However, it is very interesting to report that RLC has selected only 46 different features among the 1080 possible ones. Actually, the algorithm has produced 47 perceptual classes, each one of these corresponding exactly to one cell, allowing it to produce here again the optimal policy.

Large Gridworld with Rotations: This third setup is the same as Large Gridworld, but each time the sensors take a picture of the object, its point of view is randomly chosen in the interval $[0°, 45°]$, which adds noise to the task. RLC succeeded after 71 iterations in computing a classifier that distinguishes between 343 perceptual classes by testing 171 different features out of a set of 1141 possible features. This classifier is fine enough to obtain an optimal policy for the task.

[2] http://www.cs.columbia.edu/CAVE/coil-100.html

Figure 1. Small Gridworld topology. Cells with a cross are walls, and the exit is indicated by a gray background. Empty cells are labeled by a picture, in which circles indicate the interest points where features are extracted.

These experiments indicate the soundness of our approach. We conclude by noticing that the distribution of the images has only a marginal influence on the results.

5 Conclusions

We have introduced a new Reinforcement Learning algorithm that is able to focus the attention of the agent on distinctive and robust parts of the inputs. The abstraction level upon which RL is applied is therefore raised, since it allows the use of context-dependent information. The definition of the algorithm does not rely on a particular RL algorithm, as long as it converges under state aggregation. The pertinent features are incrementally selected in a fully autonomous, interactive fashion. This is done through a statistical analysis of the corrections that Q Learning would apply on an optimal Q function obtained through the aliased classifier. Importantly, this selection process is task-driven: Different tasks will not necessarily lead to the selection of the same subset of features.

The area of possible applications is wide, since nowadays autonomous agents are often equipped with noisy or visual sensors. For example, one of our long-term goals is to construct a robotic hand that is able to automatically learn to use optical sensors to grasp objects. However, many questions are still open. For example, it is not clear how RLC could be applied on continuous output spaces, which is indeed required for realistic robotic applications. It would also be interesting to add post-pruning operations to get rid of selected features that are subsequently proved useless for the task, and that generate overfitting effects.

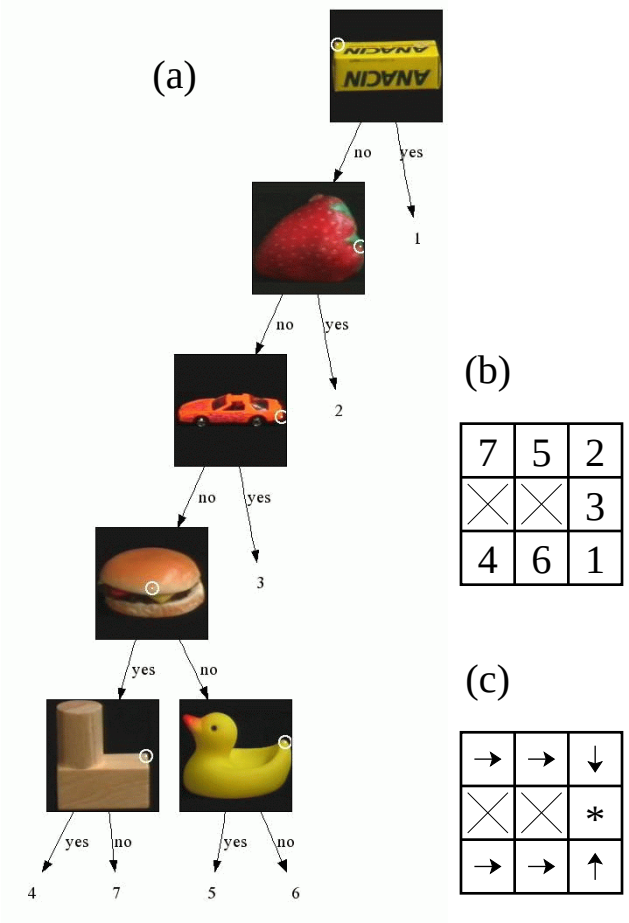

Figure 2. Resolution of the small Gridworld: (a) The last classifier $\mathcal{C}_k$, which tests the presence of the circled visual features, (b) the number of the perceptual class that is assigned to each empty cell by $\mathcal{C}_k$, and (c) the computed optimal policy for this classification, i.e. $\mathrm{argmax}_{a \in A} Q_k^*(s,a)$.

Figure 3. Large Gridworld topology (with 47 empty cells).

The study of how features can be geometrically combined to produce higher-level features that are more robust to noise [12] is another interesting research project.

References

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, 1996.

[3] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *National Conf. on Artificial Intelligence*, pages 183–188, 1992.

[4] G.J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, June 1999.

[5] V. Gouet and N. Boujemaa. Object-based queries using color points of interest. In *IEEE Workshop on Content-Based Access of Image and Video Libraries*, pages 30–36, Kauai, Hawaii, USA, 2001.

[6] M. Huber and R. Grupen. A control structure for learning locomotion gaits. In *7th Int. Symposium on Robotics and Applications*, Anchorage, May 1998.

[7] S. Jodogne and J.H. Piater. Interactive selection of visual features through reinforcement learning. In *Proc. of the 24th SGAI Int. Conf. on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge (UK), December 2004. Springer-Verlag. To appear.

[8] R.A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, NewYork, 1996.

[9] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 257–263, Madison, Wisconsin, June 2003.

[10] T.M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[11] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine learning*, 49(2-3):161–178, 2002.

[12] J.H. Piater. *Visual Feature Learning*. PhD thesis, Computer Science Department, University of Massachusetts, Amherst, MA, February 2001.

[13] J.R. Quinlan. The effect of noise on concept learning. In *Machine Learning: An Artificial Intelligence Approach: Volume II*, pages 149–166. Kaufmann, Los Altos, CA, 1986.

[14] S.P. Singh, T. Jaakkola, and M.I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. The MIT Press, 1995.

[15] R.S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proc. of 7th Int. Conf. on Machine Learning*, pages 216–224, San Mateo, 1990.

[16] J.N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[17] C.J.C.H. Watkins and P. Dayan. $Q-$learning. *Machine learning*, 8:279–292, 1992.

[18] S.D. Whitehead and D.H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991.