24th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence
Research and Development in Intelligent Systems XXI, Session 5:, Pages 285-298
The original publication is available at www.springerlink.com.

Interactive Selection of Visual Features through Reinforcement Learning

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

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

Abstract

We introduce a new class of Reinforcement Learning algorithms designed to operate in perceptual spaces containing images. They work by classifying the percepts using a computer vision algorithm specialized in image recognition, hence reducing the visual percepts to a symbolic class. This approach has the advantage of overcoming to some extent the curse of dimensionality by focusing the attention of the agent on distinctive and robust visual features.

The visual classes are learned automatically in a process that only relies on the reinforcement earned by the agent during its interaction with the environment. In this sense, the visual classes are learned interactively in a task-driven fashion, without an external supervisor. We also show how our algorithms can be extended to perceptual spaces, large or even continuous, upon which it is possible to define features.

1 Introduction

Reinforcement Learning (RL) is a general framework for modeling the behavior of an agent that learns how to perform its task through its interactions with the environment [2, 7, 22]. The agent is never told what action it should take; rather, when it does a good or a bad action, it only receives a reward or a punishment, the *reinforcement*. Schematically, RL lies between supervised learning (where an external teacher gives the correct action to the agent) and unsupervised learning (in which no clue about the goodness of the action is given). RL has had spectacular applications, e.g. turning a computer into an excellent backgammon player [23], or making a quadruped robot learn walking progressively without any human intervention [6].

In RL, the agent operates by repeating the following sequence of operations: at time t, (i) it senses its inputs in order to determine the current state s_t of the

*Research Fellow of the Belgian National Fund for Scientific Research (FNRS).

environment, (ii) it selects an action a_t, (iii) it applies this action, which results in sensing a new state s_{t+1} while perceiving a numerical reinforcement $r_{t+1} \in \mathbf{R}$, and (iv) it possibly updates its control law using this new experiment. Initially, since the agent knows nothing about what it should do, it acts randomly. After some trial-and-error interactions, the agent begins to learn its task and performs better and better. Two major challenges in RL are the exploration-versus-exploitation dilemma (should the agent exploit its history or try new actions?) and the delayed-reward problem (the pertinence of an action can appear a long time after the interaction, for example in the game of chess).

In this article, we consider the applicability of RL when the agent is faced with visual inputs. As an example, consider the task of grasping objects. It has been shown that infants learn to pre-shape their hands using their vision before they reach the object to grasp [10]. Once the contact is made, haptic feedback is used to locally optimize the grasp. For this grasping procedure to succeed, infants have to learn to distinguish between objects that require different hand shapes. Thus, infants learn to recognize objects following the needs of the grasping task. More generally, evidence shows that visual learning is task-driven [20]. Our long-term goal is to create an artificial system that would acquire object recognition skills using only its interactions with the environment [15]. RL is one plausible framework to model such a system.

Unfortunately, RL algorithms are subject to the curse of dimensionality, i.e., they are very sensitive to the number of states and actions. Now, the size of perceptual domains containing visual inputs are exponential in function of the size of the images. On the other hand, since the number of interactions an agent has at its disposal to learn its task is necessarily finite, generalization abilities are necessary to face continuous input and/or output spaces: similar perceptions are indeed expected to require similar actions. But, a robotic hand learning to grasp objects has a continuous action space.

In order to deal with these two issues, some authors have recently tried to take advantage of supervised learning techniques in the context of RL [4, 14]. Their main argument is that supervised learning comprises a large number of powerful techniques tackling high-dimensional problems with excellent generalization performances. Sketchily, these approaches reduce the RL problem to a sequence of supervised regression problems, each approximating the value of taking, in each state, any possible sequence of actions of a fixed length (the further along in the regression sequence, the greater the considered length). Using the terminology of Ernst et al. [4], we will refer to such techniques as *Fitted Q Iteration*. It seems thus promising to use Fitted Q Iteration in RL problems involving camera sensors by applying the regression algorithms directly to the values of the raw pixels. To the best of our knowledge, no material has been published on this topic yet.

Nevertheless, if Fitted Q Iteration is used on visual perceptual spaces, the embedded supervised learning algorithm will necessarily have to distinguish between visual inputs. In this sense, the learning algorithm will have to solve simultaneously a computer vision problem (image classification) and a RL prob-

lem (construction of an optimal control law). Now, it is widely admitted that vision problems are difficult to solve, and a large number of non-trivial, powerful techniques devoted to the visual recognition of objects have been developed during the last decades.

Our basic idea is therefore to facilitate the RL process by taking advantage of specialized image classification algorithms, while letting the supervised learning algorithm focus on the control law computation. Obviously, we expect that replacing the visual input by a symbolic input (i.e., the class number corresponding to the image) will drastically reduce the size of the perceptual space, and will break to some extent the curse of dimensionality.

It is clear that the idea of using vision algorithms to make RL easier is not limited to Fitted Q Iteration: actually, any RL algorithm could benefit from visual recognition. Therefore, our technique should remain general enough not to rely on a particular RL algorithm.

2 Reinforcement Learning

2.1 Markov Decision Processes

Reinforcement Learning problems are most often defined in the *Markov Decision Processes* (MDP) framework. This basically amounts to saying that, after doing some action a_t in some state s_t of the environment, the next state s_{t+1} does *not* depend on the entire history of the system, but only on s_t and a_t. This also implies that the environment obeys a discrete-time dynamics.

According to the conventions of Kaelbling et al. [7], a MDP is a tuple $\langle S, A, r, \mathcal{T} \rangle$, where S is the finite set of possible states in the environment; A is the finite set of possible actions; $r : S \times A \mapsto \mathbf{R}$ is the *reinforcement function* giving for each state-action pair the immediate reinforcement for doing this action in this state; and $\mathcal{T} : S \times A \times S \mapsto [0, 1]$ is the *transition function* giving the probability of reaching one state after doing some action in some state. Formally:

$$\mathcal{T}(s, a, s') = \mathrm{P}\left\{s_{t+1} = s' \mid s_t = s,\ a_t = a\right\}.$$

2.2 Optimal Policies and Q-functions

A *stationary Markovian control policy* (for shortness, a *policy*) is a probabilistic mapping from the states to the actions. A policy governs the behavior of the agent by specifying what action it should take in each state. RL is concerned with the construction of an optimal policy, in a sense that remains to be defined.

The goal of the agent is not to maximize its immediate reinforcements (the sequence of r_t), but its rewards over time. This leads to the definition of the *discounted return*. Given an infinite sequence of interactions, the discounted return at time t is defined by:

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

where $\gamma \in [0, 1]$ is the *discount factor* that gives the current value of the future reinforcements[1]. This means that a reward perceived k units of time later is only worth γ^k of its current value.

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: $Q^\pi(s, a) = \mathrm{E}_\pi \{R_t \mid s_t = s,\ a_t = a\}$, where E_π denotes the expected value given that the agent follows the policy π. Dynamic Programming theory [1] shows that all the optimal policies for a given MDP share the same Q function, denoted Q^*, that always exists and that satisfies the so-called Bellman's optimality equation:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} \mathcal{T}(s, a, s') \max_{a' \in A} Q^*(s', a'), \tag{2}$$

for all $s \in S$ and $a \in A$. When the Q^* function is known, for example by solving the non-linear system of Equations (2), an optimal deterministic policy π^* is easily derived by letting $\pi^*(s) = \mathrm{argmax}_{a \in A} Q^*(s, a)$ for each $s \in S$.

2.3 Overview of RL Algorithms

RL algorithms can be roughly divided in two categories: *incremental* and *batch*. In incremental RL, the agent starts with an initial policy, which is continuously updated after each interaction with the environment until convergence to an optimal policy. The popular *Q-learning* algorithm [24] belongs to this category, as well as *Sarsa* [22].

On the contrary, in batch RL, the learning process is split in two parts: (i) collection of a database of interactions, and (ii) computation of an optimal policy. The database simply contains the tuples $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ encountered during the interactions, which summarize the entire history of the system (indeed, the time information t does not matter because of the Markovian nature of the environment). *Value Iteration* and *Policy Iteration* [1] are batch RL algorithms, as well as Fitted Q Iteration (cf. Introduction). Batch RL is an interesting method when the cost of the experiments is expensive, which is the case in many robotic applications, for example grasping. It is indeed sufficient to collect once and for all a representative set of interactions.

2.4 Perceptual Aliasing

So far, we have implicitly supposed that the agent is able to distinguish between the states of the environment using only its sensors. If this is the case, the perceptual space is said *fully observable*, and the right decisions can always be made on the basis of the percepts. If it is not the case (i.e., if the perceptual space is only *partially observable*), the agent cannot distinguish between any pair of states and thus will possibly not be able to take systematically the

[1] In practice, γ is often supposed to be less than 1 to ensure the convergence of the sum.

right decision. This phenomenon is known as the *perceptual aliasing* (or *hidden state*) problem, and is closely related to ours, as it will soon become clear.

Two solutions to this general problem have been proposed in the literature: either the agent identifies and then avoids states where perceptual aliasing occurs [25], or it tries to build a short-term memory that will allow it to remove the ambiguities on its percepts [3, 9].

In this paper, we will only consider fully observable perceptual spaces. However, until the agent has learned the visual classes required to complete its task, visual percepts needing different reactions may be mapped to the same class, thus introducing perceptual aliasing. Nevertheless, the previous approaches are irrelevant in our context, since these ambiguities can be removed by further refining the image classifier. Actually, previous techniques tackle a *lack of information* inherent to the used sensors, whereas our goal is to handle a *surplus of information* related to the high redundancy of visual representations.

3 Image Classification using Visual Features

Besides RL, image classification is the other tool required by our algorithms.

The goal of image classification is to map an image to a class of objects. Recent successes in visual object recognition are due to the use of *local-appearance* approaches [8, 15, 18]. Such approaches first locate highly informative patterns in the image and in a picture of the object to be recognized, using *interest point detectors* [19], then match these interest points using a local description of their neighborhood, called a *visual feature*[2] [11]. If there are enough matches, the image is taken as belonging to the object class. As visual features are vectors of real numbers, there exists an unbounded number of features.

Local-appearance methods can deal with partial occlusions and are very flexible, since they do not need a 3D model of the objects that is frequently hard to obtain, especially for non-rigid objects. Furthermore, they take advantage of more and more powerful interesting point detectors and local descriptors.

4 Reinforcement Learning of Visual Classes

4.1 Description of our Learning System

As discussed in the Introduction, we propose to introduce an image classifier before the RL algorithm itself. The resulting architecture will be referred to as *Reinforcement Learning of Visual Classes*, and is schematically depicted at the right of Figure 1. This two-level hierarchy can be thought of as a way to raise the abstraction level on which the RL algorithm is applied: the classifier translates a low-level information (the raw values of the pixels) into a high-level information (an image class) that will itself feed the RL algorithm.

The key idea in RL of Visual Classes is to focus the attention of the agent on a small number of very distinctive visual features that allow the agent to reason

[2]The terminology "visual feature" is used here as a synonym for "local descriptor".

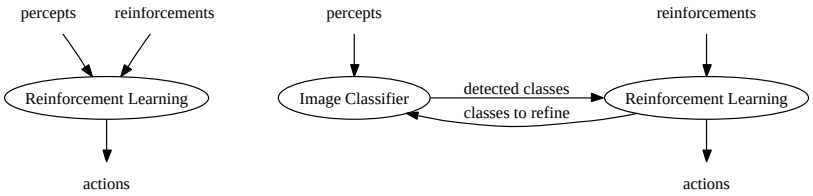

Figure 1: Comparison of information flows between "classical" Reinforcement Learning (left) and Reinforcement Learning of Visual Classes (right).

upon visual classes rather than raw pixels, and that enhance its generalization capabilities [15]. Initially, the system knows only about one class, so that all the percepts are mapped to this class. Of course, this introduces a kind of perceptual aliasing, though the perceptual space is fully observable. The challenging problem is therefore to *refine a visual class dynamically* when the agent identifies inconsistencies in the earned discounted returns when faced with that class. For instance, if the same action leads sometimes to a reward, and other times to a punishment, there is strong evidence that the agent is "missing something" in the percepts corresponding to the class. This explains the presence of a right-to-left arrow in Figure 1: the RL algorithm has to inform the image classifier when the learning of a new visual class is required.

Since there is no external supervisor telling the agent when a refinement is needed, our algorithm can only rely on statistical analysis involving the reinforcements earned by the agent. The agent will consequently learn visual classes only through interactions, which is the central property of this system. Intuitively speaking, the role of the agent is to identify *functionally-distinguishable percepts*: it should distinguish between percepts that involve different discounted returns when it chooses the same reactions.

In the sequel, we will discuss the two major elements that are required in order to turn this learning structure into a working algorithm, namely: (i) a robust criterion able to decide when the classification is not fine enough, that will be called the *aliasing criterion*, and (ii) an image classifier able to refine a class on request by learning a new distinctive visual feature. To conclude this general description, note that the visual features should be powerful enough to distinguish any functionally-distinguishable percept. We will suppose that this weak requirement is met in the rest of the paper.

4.2 Detailed Description

4.2.1 Core of the Algorithm

The previous section has introduced the paradigm of Reinforcement Learning of Visual Classes. We are now ready to give an in-depth view of our algorithm, which operates in batch mode, since it relies on a statistical analysis of the

discounted return observed during the interactions. Here is its core:

1. Begin with step count $k := 0$ and a percept classifier $\mathcal{C}_k$ that maps all the percepts to a single class, i.e., such that $\mathcal{C}_k(s) = 1$ for all the percepts s;
2. Collect a database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}, e_t \rangle$, where s_t are the raw percepts[3] furnished by the sensors, and e_t is a Boolean tag indicating whether the action a_t has been chosen by randomization or by deterministic exploitation of the knowledge of the agent from the previous steps;
3. After N interactions have been collected:
   (a) Use the aliasing criterion to decide if a class needs to be refined,
   (b) While there exist aliased classes, refine the classifier $\mathcal{C}_k$ by learning new distinctive visual features, which leads to a new classifier $\mathcal{C}_{k+1}$,
   (c) Let $k := k + 1$. If k is below some threshold value, go to Step 2.
4. Use a RL algorithm to control the system through the last classifier $\mathcal{C}_k$.

The way the interactions are acquired at the second step is unimportant. For example, a simple ε-greedy policy can be used in order to collect the database. Nevertheless, the database has to satisfy the following requirement, the reason of which will be explained in the next section: at a given step k, whenever the agent chooses to deterministically exploit its knowledge at some time t, it should do so for at least the next k interactions, i.e., up to time $t + k$.

4.2.2 Aliasing Criterion

The only information available to the aliasing criterion is given by the reinforcement values present in the database of interactions. Since the database is necessarily finite, our criterion can only rely on an approximation of the discounted returns (see Equation (1)) observed in the database over some *finite* time horizon. This leads to the following definition:

Definition 1 *The* truncated discounted return *at some time t for a time horizon H is defined as $\hat{R}_t^H = \sum_{i=0}^{H} \gamma^i r_{t+i+1}$, where r_{t+i+1} are the reinforcements present in the database. It is left undefined if t is greater than $N - H$.*

Let us now suppose that the environment is deterministic, i.e., the transition function $\mathcal{T}$ of the underlying MDP is deterministic. In this context, it is clear that executing the same sequence of actions starting from a given state will always lead to the same truncated discounted return. Therefore, two states can be distinguished using only the reinforcements if *there exists some sequence of actions* such that executing this sequence in those two states leads to different truncated discounted returns. Of course, we cannot try every possible sequence of actions, so we restrict ourselves to the sequences present in the database.

[3]Here, s_t denotes at the same time a percept and a state. This syntax is justified since the perceptual space is fully observable: there is a mapping from the percepts to the states.

Now, there could be random variations in the truncated discounted returns just because of the non-deterministic nature of the exploration policy. Such variations should obviously not be taken into account in the aliasing criterion. This explains the requirement on the database of interactions introduced at the end of Section 4.2.1: by considering only the sequences of actions starting in states marked as obtained from deterministic exploitation of the system history, which can be determined by testing the flag e_t, we ensure the uniqueness of the considered sequences of actions.

At some step k of our algorithm, this uniqueness is only ensured for sequences of actions of length less than k. The aliasing criterion is thus based upon an incremental construction: at the step k of our algorithm, we only try to distinguish states that are aliased by considering sequences of actions of length k present in the database, i.e., that are k-aliased. More formally:

Definition 2 *Two states s_t and $s_{t'}$ belonging to the same visual class (i.e., such that $\mathcal{C}_k(s_t) = \mathcal{C}_k(s_{t'})$) and encountered respectively at times t and t', are* k-aliased *if they have both been tagged as obtained from deterministic exploitation, and if $\hat{R}^k_t \neq \hat{R}^k_{t'}$.*

Of course, the more interactions are collected, the more fine-grained distinctions between states can be discovered. Note that the number of iterations of our algorithm corresponds to the maximum time horizon to consider.

4.2.3 Class Refinement

The class refining operation has to discover a new visual feature that best explains the variation in the truncated discounted returns for some visual class at some time horizon k. This is a classification problem, for which we propose a variation of the standard splitting rule used when building decision trees [12].

Firstly, we sort the observed truncated discounted returns obtained starting from the considered class. Each cutpoint in the obtained sequence induces a binary partition of the visual percepts mapped to this class: the percepts such that the corresponding truncated returns are above the cutpoint, and the others. Then, for each possible cutpoint, we extract the visual feature that maximizes some information-theoretic score for this partition into two buckets of visual percepts. This is done by iterating over all the visual features present around the interest points in the considered percepts, that are in finite number, and evaluating the split induced by each one of those features. We finally keep only the visual feature that has the maximal score among all the extracted visual features.

4.2.4 Non-deterministic Environments

We have supposed since Section 4.2.2 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 visual feature induces a genuine split that is significantly different from a random split. This approach is inspired from decision tree pruning [17].

4.3 Using Decision Trees as Classifiers

The concrete classifier used in our implementation has not been discussed yet. In this work, we have been working with binary decision trees: the visual classes correspond to the leaves of the tree, and the internal nodes are labeled by the visual feature, the presence of which is to be tested in that node. The classification of a percept consists in starting from the root node, then progressing in the tree structure according to the presence or the absence of each visual feature found during the descent, until reaching a leaf.

To refine a visual class using a visual feature, it is sufficient to replace the leaf corresponding to this class by an internal node testing the presence or the absence of this feature, and leading to two new leaves.

5 Reinforcement Learning of Classes

The approach we have just presented is actually not limited to visual inputs. It could indeed be useful in any perceptual domain (possibly continuous) that supports classification as a way to reduce its size. In this context, the "visual features" would become "features", i.e., properties that can be displayed or not by the raw percepts. For example, a feature could be the value of a bit in the case of percepts containing binary numbers. Our technique could also be applied for agents having noisy sensors: the use of distinctive features would allow the agents to get rid of noise by examining only pertinent and robust parts of their percepts.

All the previous algorithms can readily be adapted to perceptual spaces upon which the following three elements can be defined:

Features: A feature is any property a raw percept can exhibit or not. There can possibly be an infinite number of features.

Feature Detector: It is a function that tells whether or not a given raw percept exhibits a given feature.

Refining Oracle: It is an oracle that, given two sets of raw percepts, returns the most informative feature explaining this partition into two subsets. It is introduced as an oracle since it is allowed to use some context-dependent information to direct the search of the best feature: the oracle is not obliged to exhaustively consider every feature, which makes a particular sense when there is an infinite number of features.

We will call such a generalization *Reinforcement Learning of Classes*.

6 Experiments

We have investigated the behavior of Reinforcement Learning of (Visual) Classes in the context of a simple navigation problem, namely escaping from a discrete 2D maze constituted of empty cells and walls. The goal of the agent

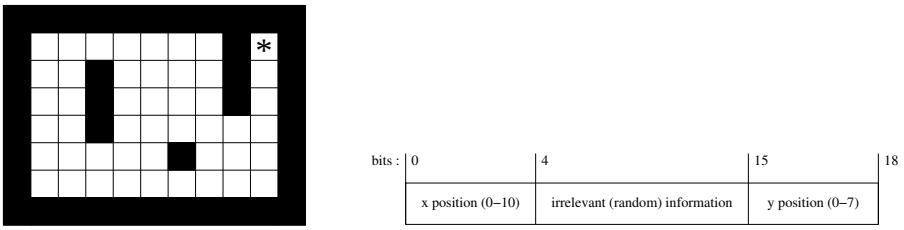

Figure 2: On the left, Sutton's Gridworld [21]. Filled squares are walls, and the exit is indicated by an asterisk. On the right, a diagram describing the percepts of the agent, that are binary numbers of 18 bits.

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. When a move would take the agent into a wall, the location is not changed. When a move takes it into the exit, the agent is randomly teleported elsewhere in the maze. The agent earns a reward of 100 whenever 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" [21], 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: in a first experiment, the percepts will be binary numbers that contain the binary values of x and y; in a second experiment, a different object will be buried in each cell under a transparent glass, and the sensors of the agent will return a picture of the object underneath.

6.1 The "Binary" Gridworld

In this first experiment, we have used the original Gridworld topology, which is depicted at the left of Figure 2. The sensors of the agent return a binary number, the structure of which is shown on the right of the same figure. In this experiment, features are defined as the bits of the binary numbers, so RL of Classes has been applied. Here, the feature detector tests if a given bit is set or not, and the refining oracle seeks the most informative bit explaining the partition into two subsets of binary numbers.

To achieve its task, the agent has to focus its attention on the bits encoding x and y, since the other bits are random, and thus irrelevant to its task. We have noticed that this is indeed the case: the built classifier only uses the bits 0, 1, 2, 3, 15, 16 and 17. The obtained classification is shown in Figure 3, as well as the optimal policy it involves. It can easily be seen that the built policy is optimal. After k has reached the value 15, which roughly corresponds to the diameter of the maze, no further split was produced. Note however that this value can vary depending on the database of interactions collected.

It is important to notice that the classification rule is obtained without pre-treatment, nor human intervention. The agent is initially not aware of

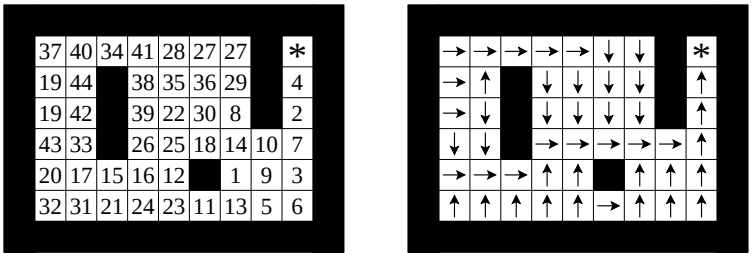

Figure 3: On the left, the classification using bits obtained at the end of our algorithm. On the right, the policy built using the last classifier.

which bits are important. Moreover, the interest of using features is clear in this application: a direct tabular representation of the Q function would have $2^{18} \times 4$ cells (one for each possible pair of a binary number and an action).

6.2 The "Tiled" Gridworld

The goal of this second experiment is to illustrate RL of Visual Classes on the toy example depicted in Figure 4. The navigation rules are identical to the Binary Gridworld, but there are fewer cells in order to better interpret the results. The percepts are color images of objects taken from the COIL-100 database [13]. Each cell is identified by a different object. The used visual features are color differential invariants detected at Harris color points of interest [5].

Figures 4 and 5 depict the obtained results. The algorithm succeeds at distinguishing between visual inputs requiring different reactions. Once k has reached the value 3, no further refinement has taken place. It is interesting to notice that the hamburger and the wooden toy (class 4), as well as the duck and the boat (class 5), have not been distinguished. This is a desirable property since these states require the same action, i.e. to go right.

7 Conclusions

We have introduced algorithms that succeed in learning distinctive features in an interactive context, using only a reinforcement feedback. Our approach is quite general, and is applicable to many perceptual domains (large, continuous and/or noisy). In particular, these algorithms can be applied in RL problems with image inputs. The only restrictions on the perceptual space are that it must be fully observable, and that it must be possible to define features in it. To achieve this goal, techniques similar to those used in supervised decision tree construction are exploited (evaluation of splits using information-theoretic measures, and reduction of overfitting through hypothesis testing). The architecture of our system is independent of the underlying RL algorithm.

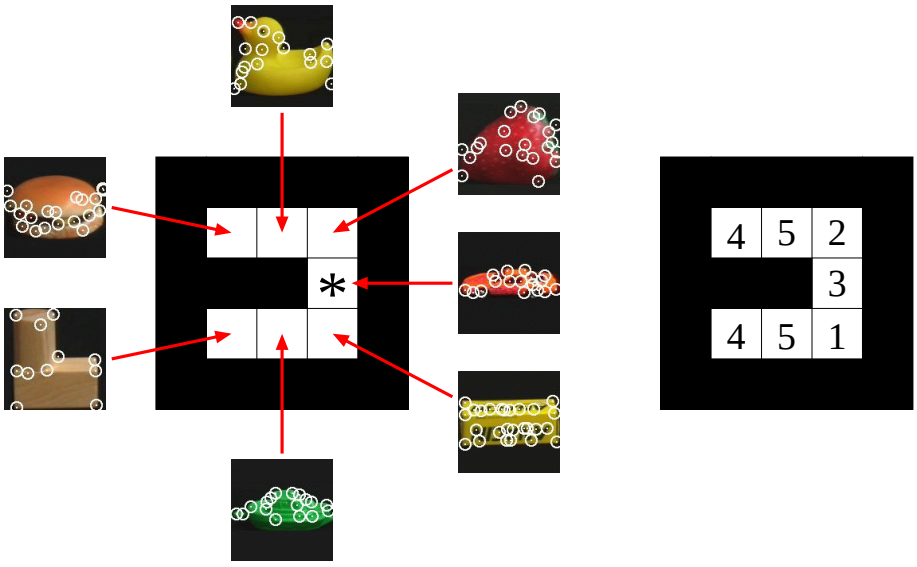

Figure 4: On the left, the Tiled Gridworld. The objects under each cell are marked with their interest points circled. The exit is labeled by an asterisk. On the right, the learned classification.

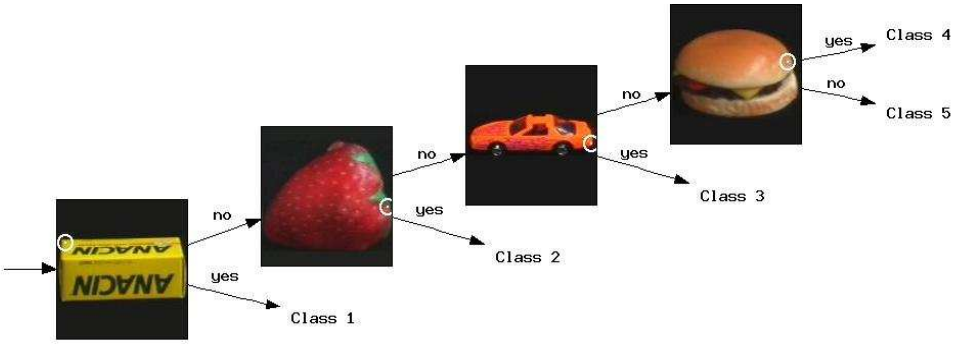

Figure 5: The resulting decision tree. The visual features tested in each internal node are circled.

Its two-level hierarchy with top-down and bottom-up information flows enables to raise the abstraction level upon with the embedded RL algorithm is applied.

Our work can be seen as a generalization of the visual feature learning system that has been applied by Piater to grasp objects [15]. Indeed, this system can only be applied in interactive tasks with no delayed reward (i.e., where $\gamma = 0$) and with binary reinforcements (i.e., only two reinforcements are possible: either good or bad action). Moreover, our work is to be distinguished from the tree-based discretization technique of Pyeatt and Howe [16], since the latter is specific to Q-learning, and since its discretization of the perceptual space relies on the perceptual values rather than on higher-level features.

Future research should try to adapt RL of (Visual) Classes to problems with continuous perceptual and/or action spaces, for example grasping. On the other hand, techniques to remove learned features that are subsequently proved to be useless could be developed. To evaluate the performance of different classifiers (e.g., naive Bayes), and to use more powerful visual features (e.g., affine-invariant features, or features taking semi-local constraints into account), are other interesting research topics.

References

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

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

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

[4] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning, 2004. Submitted for publication.

[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, AK, May 1998. TSI Press.

[7] L.P. Kaelbling, M.L. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[8] T.K. Leung, M.C. Burl, and P. Perona. Finding faces in cluttered scenes using random labeled graph matching. In *Proc. of the Fifth International Conference on Computer Vision*, page 637. IEEE Computer Society, 1995.

[9] R.A. McCallum. *Reinforcement learning with selective perception and Hidden State*. PhD thesis, University of Rochestor, Rochestor, NewYork, 1996.

[10] M. McCarty, R. Clifton, D. Ashmead, P. Lee, and N. Goubet. How infants use vision for grasping objects. *Child Development*, 72:973–987, 2001.

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

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

[13] S.A. Nene, S.K. Nayar, and H. Murase. Columbia object image library (COIL-100). Technical Report CUCS-006-96, Columbia University, New York, NY, February 1996.

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

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

[16] L.D. Pyeatt and A.E. Howe. Decision tree function approximation in reinforcement learning. In *Proc. of the Third International Symposium on Adaptive Systems*, pages 70–77, Havana, Cuba, March 2001.

[17] 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.

[18] C. Schmid and R. Mohr. Local greyvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–535, 1997.

[19] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

[20] P.G. Schyns and L. Rodet. Categorization creates functional features. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 23(3):681–696, 1997.

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

[22] R.S. Sutton and A.G. Barto. *Reinforcement Learning, an Introduction*. MIT Press, 1998.

[23] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

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

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