Exploiting qualitative knowledge to enhance skill acquisition

Cristina Baroglio
baroglio@di.unito.it
tel. +39-11-7429229 fax +39-11-751603

Dip. di Informatica, Università degli Studi
c.so Svizzera 185, I-10149, Torino, Italy

Abstract. One of the most interesting problems faced by Artificial Intelligence researchers is to reproduce a capability typical of living beings: that of learning to perform motor tasks, a problem known as *skill acquisition*. A very difficult purpose because the overwhole behavior of an agent is the result of quite a complex activity, involving sensory, planning and motor processing. In this paper, I present a novel approach for acquiring new skills, named *Soft Teaching*, that is characterized by a learning by experience process, in which an agent exploits a symbolic, qualitative description of the task to perform, that cannot, however, be used directly for control purposes. A specific Soft Teaching technique, named *Symmetries*, was implemented and tested against a continuous-domained version of well-known pole-balancing.

Keywords: skill acquisition, knowledge-based feedback, adaptive agents.

1 Introduction

In this work we consider a framework where an *artificial agent* interacts with its environment by executing actions. Our *aim* is to have the agent performing a specific *task* by generating a proper control device. Generally speaking, the controller can be built in many ways (from direct encoding to the use of AI techniques); in this paper we are interested in the approaches that exploit *on-line learning methods*. The reason is that using learning methods less information is to be supplied explicitly to the agent, its lack being balanced by the capability of acquiring it autonomously, which simplifies the controller's production.

In the literature, learning rules have already been used to acquire control knowledge either from examples, e.g. [13], or from the agent's experience, e.g. [14, 6]; this latter solution is known as *adaption*. In order to have adaption, one must find a form of *feedback*, that can be "processed" by the agent's learning rule. Typically, adaption is obtained by means of *Reinforcement Learning* (RL). As the name suggests, however, the feedback returned in this case is quite poor: in fact, it is a scalar value, corresponding to a reward when the agent satisfies the goal condition or to a punishment when it enters a failure state. Then, in RL the focus of attention is mostly posed on how to learn a policy given that the states of the underlying model are known, a hypothesis that does not hold in most real-world tasks. To overcome this limit, RL was applied to agents implemented as *neural networks*, but so, convergence cannot be guaranteed anymore [12].

One feature that is often overlooked when dealing with adaption is that live agents (such as us) exploit a lot of *knowledge* to help their learning. We do not know how this knowledge is "encoded" in the brains but we know that we use it to monitor our behavior, to recover from errors, to describe in words a task we want to have accomplished and use such simple descriptions to guide its execution. This is the focus of interest of this work.

2 Soft Teaching

The *Soft Teaching* methodology is based on a particular use of knowledge that can be observed in our everyday experience. Let's see it with an example. Supposing a human teacher has to explain the pole-balancing task to a human learner, (s)he would probably say something like "Manouver the cart, moving it left or right, so to keep the pole on it vertical". Note that this description does not contain any suggestion of what to do, "moving it left or right" is just the list of the possible actions that can be taken. We say that this description is *non-operational* because the learner still has to find a policy that allows him/her to achieve the goal. By using the above knowledge, however, the learner can criticize his/her behavior during training. We call such a mechanism the *Human-Teacher-Human-Learner* (or HTHL) interaction scheme: since the teacher cannot transfer his/her skill to the learner, it transfers some information that (1) makes the learner *aware* of what is to be achieved and (2) allows him/her to *start* and *bias* a learning process that leads to building some new actuative knowledge.

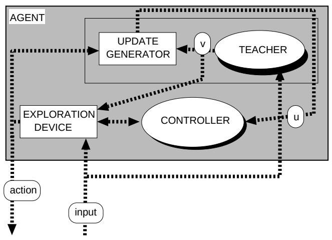

Fig. 1. The proposed architecture.

The architecture presented here (Fig. 1) is based on the HTHL scheme: the adaptive agent owns some *qualitative knowledge* (given in a *symbolic* notation) about the task and the domain, that can be turned into a very precise evaluator. The knowledge is supplied by a human teacher, who does not take part to the learning process, and is *non-operational*. The module that handles the qualitative knowledge is called *teacher* or *teacher-on-line* It is the "trait d'union" between the *symbolic* level, to which knowledge belongs, and the *analogical* level of the controller. In this framework, an adaptive agent can be trained by producing different kinds of *feedback signals* and/or using different *training algorithms*.

Indirect Supervised Learning In particular, this work introduces a subset of Soft Teaching methods called *Indirect Supervised Learning* (or simply ISL) methods. A training technique belongs to the ISL family if (1) it exploits a *supervised* learning rule and (2) the error is computed as a function of the difference between the action built by the controller and an alternative action generated by the *teacher-on-line*. This latter feature allows training to be performed *on-line*, keeping, on one hand, the advantages related to learning by interaction (of classical RL) and, on the other, overcoming some of the limits of the approaches to teaching that are already in the literature (Section 4). *Indirect Supervised Learning* is, then, a new learning paradigm in which the teacher returns *suggestions* in addition to reinforcements. The approach is somewhat similar to the one proposed by Clouse and Utgoff [4] (see Section 4), the difference is that in our case the decision of *when* and *what to suggest* is taken *automatically* instead than by a human teacher (that is what "indirect" stands for).

From an abstract perspective, the ISL *teacher architecture* is made of three main modules: a *strategy generation module*, a *strategy evaluation module*, and a *model of the world*. The main loop of the ISL algorithm is as follows:

1. given a world situation and current controller, build a strategy S;
2. produce a set of alternative strategies to the one built in step 1;
3. evaluate each strategy using the model and the teaching knowledge;
4. apply the best scoring strategy S';
5. if the effects are as those predicted by the model, update the controller exploiting the applied strategy S' and the original strategy S; otherwise, update the model in a supervised way;
6. if the trial is not over go to step 1.

The idea of exploiting a world model is not new in control applications (see [10]), the real novelty stands in the way strategies are evaluated. All the strategies taken into account in the learning process are built in one of the two following ways. The first consists in getting a situation from the world, using the current controller to produce a first action and, at last, entering a loop in which, first, the next situation is predicted by the model, then, the controller is used to generate next action according to the prediction. Strategy's length is defined by the human teacher. The model of the world is intended to be a *learning* module. So far, however, in order to study the effect of ISL in skill acquisition avoiding any additional complication, a *perfect* model was used. Once a strategy is available, it is possible to produce a set of alternatives by modifying it so to investigate the surroundings of the search space. Of course, different methods can be applied to this aim. In the implementation, random variants were used.

Symmetries *Symmetries* is an ISL method, used to better exploit the information contained in the best-scoring strategy. The idea is simple: in tasks like pole-balancing the domain is naturally symmetric, i.e. in absolute values, what the controller should do when the pole falls at its right is exactly what it should do when it falls at its left. A human learner realizes such a structure in the domain and exploits it. The same should happen in artificial learners.

Symmetries goes beyond the current perceptions of the agent, elaborating its experience. Such an elaboration is to be done *on-line* because the only alternative is to accumulate long traces of execution to post-process. The method developed alternates experience to *hypothetical reasoning*: once S' was applied and the controller updated, a subset of the points that are symmetric (w.r.t. 0) to the state that caused S''s generation are built (in the experiments on pole-balancing only x and θ signs are changed). The strategy that the agent would apply starting from each of them is built together with a proper symmetric version S'_i of S'. Both are evaluated by the teacher-on-line: if S'_i scores better than the strategy produced by the controller, it is used as a desired target by the update rule. The check is necessary because the current controller is not bound to be symmetric.

Strategy evaluation Strategies are evaluated by means of the teacher's *knowledge*. The first step is to give it a *spatial interpretation* so to allow *scalar* evaluations. A thorough description of this process can be found in [1]. Then, a single action can be evaluated by mapping on this space the input situations before and after the action was applied. Given an input $x1$ and a predicate P with a fuzzy semantics, we define P's *distance from truth* as follows: distFromTruth($P(x1)$) is 0 if $P(x1)$ *is* true, the distance between $x1$ and the truth set of P otherwise. For a better understanding let's consider predicate *closeToEnd*, whose semantics is as follows: $closeToEnd(x) \Leftrightarrow distance(x, END) < threshold$. $closeToEnd(x1)$ returns $threshold - distance(x1, END)$, where END is a given constant. Distance from truth can be extended to conjunctions of predicates by using a Euclidean-like distance measure, e.g., if $goal = P_1$ && ... && P_n, $distFromTruth(goal(x1))$ will be $\sum_{i=1,\ldots,n} distFromTruth(P_i(x1))^2$. When the sum returns a value different than 0 (condition not satisfied) the *translated sigmoid function* $(1 + exp^{-\alpha x})^{-1} - 1$ is used to map it on the range $[-1, 0]$. The negative interval accounts for the fact that the evaluations can be used directly as negative reinforcements. Last, if the direction of movement is different than the desired one, the evaluation is discounted.

In order to evaluate a strategy the evaluations of each single step are combined by means of a discounted sum $eval_n = goodness(a_n) + \gamma \; eval_{n-1}$ where a_n is the action taken at time-step n and $\gamma \in [0, 1]$ is a discount factor.

The learner In the experiments, the learner was implemented as a Neural Network: this choice does not restrict the set of functions that can be approximated because it is well-known [7] that a three-layered neural network can approximate any function. Different models were tried; the ones that gave the best results belong to the *Locally Receptive Fields* family (or LRFNs, see [2]). They perform a piecewise approximation, being each hidden neuron's activation region a closed area of the input space. Then, every update has a *local* effect, leaving the knowledge acquired for situations far from the current one unchanged. This *locality* turned out to be very important to overcome *unlearning*, to find conditions that allow to check if the hidden neurons number is sufficient to represent the desired control function, and to dynamically grow the network during the learning process (see [10, 5]). The networks were implemented by means of the Fuzzy-Neural system already described in [2]: a same network topology allows to implement, alternatively, *Fuzzy Controllers* (FCs) and *Radial Basis Function Networks* (RBFNs). A network can be interpreted as a fuzzy knowledge base and the learning process preserves the rules' structure. In all experiments the same initial controller, made of 25 random rules, was used.

```
[region]
    test1      =  (THETA > 0);
    goal       =  (inclination() == 0);
    failure    =  ((THETA > 0.2094384) || (X >= 2.4));
[context1]
    X          =  any;
    THETA      =  less;
[static-cond1]
    fluxpipe   =  (vertical() && keep - off());
```

3 Experimental results

The method described in this paper was tested against *pole-balancing* [1]. The reason is that it is one of the most classical RL testbeds; it is well understood and allows comparisons with other techniques. Furthermore, although it is simpler than most real-world tasks, it allows to face many of the difficulties shown by continuous-domained control problems. The pole-balancing simulator is, depending on the experiment, exactly the one made freely available by Rich Sutton[1] (we will call it RS, for short) or a variant of it that can handle continuous actions. In all experiments the controller has the same four inputs $\langle X, \dot{X}, \theta, \dot{\theta} \rangle$: the position of the cart, its time derivative, the inclination of the pole, and its time derivative. The goal is to keep the pole balanced for 100,000 timesteps. For each experiment, the generalization ability of the learned controllers was checked by starting part of the trials in situations different than those used during training. Generalization tests are extremely interesting because interacting with the world, the experience is limited to the portion of the input space directly faced during the exploration. As we will see, using *Simmetries* it is possible to increase the speed of learning allowing a neural approximator to learn a continuous pole-balancing control function in *half* the number of training trials required by RS to learn a policy in which only *two* alternative actions are allowed. The learned controllers also show *very good* generalization performances. The domain knowledge is reported above. It is a coding of: *the goal is to keep the pole balanced; if you bump against the ends of the track or the pole's inclination is greater than a certain angle you fail.* It is encoded by keywords GOAL and FAILURE. The other sections are used to give more information about the behavior desired when the agent is in a particular context (see [1] for details). For the sake of simplicity only the case (THETA>0) is reported (the other being symmetric).

Experiments in the RL framework. Different experiments were carried on using a traditional reinforcement function. They were done for comparison purposes. First of all, RS was tried. The reinforcement function used by this system returns -1 in case of failure and 0 otherwise. The state space is divided in 162 boxes; for each input situation the box containing it is selected and an action is produced according to this selection (see [3]). In RS all the experiments start from situation $\langle 0, 0, 0, 0 \rangle$ and training stops after 100 trials if the goal is not reached. RS learned to balance the pole in a number of trails $n \in [58, 79]$.

Then, I checked if two classical reinforcement functions (one returning -1 for failures and 0 otherwise and the other returning -1 for failure, 1 for success and 0 otherwise) could guide a LRFN in learning to balance a pole *without* using any particular RL technique, i.e. simply using it instead of a teacher. The task is much more difficult than before because a *continuous* function is to be acquired. In both cases the LRFN shows *no* learning performance.

Last and due to the fact that in most of the experimens below the initial situation is different than $\langle 0, 0, 0, 0 \rangle$, RS was started in situations respectively equal to $\langle 0, 0, r, 0 \rangle$ and $\langle r, 0, r, 0 \rangle$, where r stands for a random value. Strangely enough, in case $\langle 0, 0, r, 0 \rangle$ the agent is not able to keep the pole balanced for more than the initial timestep. In the other case, instead, learning takes place but it takes over 100 trials two attempts out of three. At the third attempt it learns in 72 trials.

Experiments in the ISL framework. Due to strategy random generation, typical of the ISL framework, each experiment was repeated up to 5 times. In all experiments the same teaching knowledge and the same initial LRFN were used. However, depending on the antecedent composition rule chosen, the net could be either a FC or a RBFN. A few words are, now, to be spent about the

[1] ftp.gte.com.

distribution of the initial situations faced during training and test. Depending on how close a situation is to satisfying the failure condition, we can characterize initial situations by their degree of *difficulty*. Random generation affects only two input variables: x, the cart's position and θ, the pole's inclination. Since θ value is always very close to 0, the difficulty mostly depends on x. Then, for a better evaluation of the results we must take into account the distribution of the initial situations produced w.r.t. this feature.

Table 1. Symmetries off. The columns report: the parameter values (strategy length and γ); the (avg.) first successful trial; the (avg.) successful test trials starting in $\langle 0,0,r,0\rangle$; the (avg.) successful test trials starting in $\langle r,0,r,0\rangle$.

configuration	first success	$\langle 0,0,r,0\rangle$	$\langle r,0,r,0\rangle$
1 step $\gamma = 0.9$	392	0 out of 10	0 out of 10
5 steps $\gamma = 0.9$	114	4.33 out of 10	0.33 out of 10
10 steps $\gamma = 0$	308	4.5 out of 10	1 out of 10
$\gamma = 0.1$	311	0.25 out of 10	0 out of 10
$\gamma = 0.4$	348	0 out of 10	0.66 out of 10
$\gamma = 0.5$	398	8.75 out of 10	2.5 out of 10
$\gamma = 0.9$	304	7.66 out of 10	2 out of 10

To this aim, the domain $[-2.4, 2.4]$ was split in three ranges: $peril = [-2.4, -2.0] \cup [2.0, 2.4]$ $semiperil = [-2.0, -1.6] \cup [1.6, 2.0]$, and $safe = [-1.6, 1.6]$, ordered from the closest to the farthest from failure. The distribution is as follows: 73% of initial situations belong to $safe$, 16% belong to $semiperil$, and the last 11% to $peril$. Since a situation belonging to $peril$ can be non-recoverable (pole falling towards the end of the track), in the worst case (all $peril$ situations non-recoverable) the maximum performance expected is 89%; more frequently, when half of the $peril$ situations cannot be recovered a maximum 94.5% performance can be obtained. However, when the starting situation is of the same type as during training ($\langle 0,0,r,0\rangle$), a 100% performance is expected.

Simmetries was applied both to RBFNs and to FCs. In both cases, the number of trials required dropped considerably w.r.t. training without Symmetries but RBFNs showed a *stabler* performance, i.e. the speed of learning of the same initial controller in different trials all configured in the same way did not vary too much. This is why most of the experiments were done on RBFNs. Their learning behavior was incredibly good. First of all, the speed of learning was always much higher than without Symmetries dropping from about 300 trials (see Table 1) to a value in between the 20th and the 40th trial. Note that these speeds are even higher than those of RS (and let's recall that RS's learning problem is easier and that the number of rules it uses is 162 vs. the 25 rules used here). The second nice feature of the learning process was that in the case in question the pole's oscillation was always reduced very quickly to an almost unperceivable movement. During tests, the controllers did from 9 to 10 successful trials out of 10 in $\langle 0,0,r,0\rangle$ and (in the average) 6 trials out of 10 in $\langle r,0,r,0\rangle$, for an overall 96% of success in $\langle 0,0,r,0\rangle$ and 60% otherwise.

However, using RBFNs one must pay attention to limit the update to the

Table 2. Test performances of the set of controllers saved every 20,000 timesteps.

cntr. saved at timestep	avg. perf. in trial $\langle 0, 0, r, 0\rangle$	avg. perf. in trial $\langle r, 0, r, 0\rangle$
30,000	418	397
50,000	397	606
70,000	356	211
90,000	380	280

rules whose activation is high, otherwise *unlearning* may occur. The results reported above were obtained updating rules whose activation was higher than 0.4 but when the threshold is decreased things change. Table 2 shows what happens using a threshold equal to 0.2: during the first successful trial of the training phase, the controller was saved at fixed time intervals obtaining a set of different controllers that were, then, tested as in the previous experiments. All the controllers up to the 10,000th time-step show the same good performances. However, when training passes the 10,000th timestep, the controllers obtained are *no more* able to balance the pole for a long enough time and the average performance decreases a little bit as the time passes (seeTable 2). The cause is *over-fitting*: during the first successful trial, the controller specializes to return lower and lower forces as the oscillation diminishes forgetting what to do when the oscillation is greater. Note, however, that increasing the threshold too much (0.6 in the experiments) prevents the controller from learning because rules are never updated.

4 Related work

In the literature, there are few examples of teaching applied to learning skills. The first and most common way of teaching a task to an agent is to produce a learning set made of examples of good execution *recorded* by a skilled human operator. Each recorded trace is a sequence of pairs $\langle x^t, a^t\rangle$ where x^t is the input situation at time step t, and a^t is the correspondent action applied by the operator. Afterwards, a learning algorithm is applied off-line to induce a skill. This approach is commonly followed when the agent is implemented as a *neural network*. Teaching by examples has explicitly been used in the work by Kaiser [9, 8], and in the Fuzzy Neural methodology [2] and has also been applied in RL by Lin [11]. The main difference w.r.t. the aforementioned approach, is that here each sequence of pairs ends with a reinforcement value. The learner is shown the sequences backwards and uses a RL rule to update an approximation of the value function. Learning by examples shows, however, a couple of drawbacks. The first is that, due to the existing learning rules (such as gradient descent), as long as the learner works on recorded examples only, it is very difficult for it to *outperform* its teacher. Unfortunately, there are applications, especially industrial tasks, in which on one hand, it is not possible to generate examples of *optimal* behavior (the human operator is not a perfect operator); on the other, often it is not even possible to produce enough examples. An alternative consists in giving *hints* to the learner when necessary, as done by Clouse and Utgoff [4]: in the work developed so far, the teacher is supposed to be a *human operator* who monitors the learning process, which, in turn, exploits RL techniques. When the agent is stuck in a situation it is not able to solve the teacher gives it a hint, i.e.

(s)he suggests what to do. The agent applies the action and learning goes on. However, in many tasks a human expert cannot monitor the learning process of the agent because a continuous stream of inputs at a very low perceptive level (forces, speeds, ...) is to be handled quickly.

5 Conclusions

In this paper a novel approach to skill acquisition, named *Soft Teaching*, is proposed; its main characteristic is that, differently than what can be found in the literature, the teaching process is performed in a fully *automatic* way, exploiting some *qualitative knowledge* about both the goal and the domain. Such knowledge is non-operational, i.e. it cannot be used to directly control the agent, but can be used to evaluate the agent's behavior. In this sense, Soft Teaching is inspired by the described HTHL scheme. Among the many different architectures that can be developed within Soft Teaching, the focus is posed on *Indirect Supervised Learning* methods, in which Soft Teaching is used jointly with a supervised learning rule. In particular, a specific ISL technique, named *Symmetries*, was implemented and tested against pole-balancing. Experimental comparisons show that Symmetries allows to obtain very good performances reducing the number of trials necessary to learn a controller of one order of magnitude.

References

1. C. Baroglio. Teaching by shaping. In *the ICML Workshop on learning by induction vs. learning by demonstration*, Tahoe City, CA, USA, 1995.
2. C. Baroglio, A. Giordana, M. Kaiser., M. Nuttin, and R. Piola. Learning controllers for industrial robots. *Machine Learning, Spec. Iss. on Learning Robots*, (23), 1996.
3. A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning prolems. *IEEE Trans. on SMC*, SMC-13:834–836, 1986.
4. J.A. Clouse and P.E. Utgoff. A teaching method for reinforcement learning. In *Proc. of the Machine Learning Conference MLC-92*, pages 92–101, 1992.
5. B. Fritzke. Growing Cell Structures – a self-organized network for unsupervised and supervised learning. *Neural Networks*, 7(9), 1994.
6. V. Gullapalli. A stochastic reinforcement learning algorithm for learning real valued functions. *Neural Networks*, 3:671–692, 1990.
7. K. Hornik, M. Stinchcombe, and H. White. Multilayer feed-forward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
8. M. Kaiser and J. Kreuziger. Integration of symbolic and connectionist processing to ease robot programming and control. In *ECAI'94 Workshop on Combining Symbolic and Connectionist Processing*, pages 20 – 29, 1994.
9. M. Kaiser and F. Wallner. Using machine learning for enhancing mobile robots' skills. In *IRTICS-93*, 1993.
10. P. Katenkamp. Constructing controllers from examples. Master Thesis, Univ. of Karlsruhe, Germany, 1995.
11. L.J. Lin. Self-improving reactive agents based on reinforcement learning, p lanning and teaching. *Machine Learning*, 8:293–321, 1992.
12. M.A.F. McDonald. Approximate discounted dynamic programming is unreliable. Technical Report 94/6, Dept. of Comp. Sci., Univ. of Western Australia, Oct. 1994.
13. L.X. Wang and J.M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Trans. on SMC*, SMC-22(6):1414–1427, November 1992.
14. C.J.C.H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, May 1992.