Chapter I

Machine Learning for Agents and Multi-Agent Systems

Daniel Kudenko
University of York, UK

Dimitar Kazakov
University of York, UK

Eduardo Alonso
City University, UK

ABSTRACT

In order to be truly autonomous, agents need the ability to learn from and adapt to the environment and other agents. This chapter introduces key concepts of machine learning and how they apply to agent and multi-agent systems. Rather than present a comprehensive survey, we discuss a number of issues that we believe are important in the design of learning agents and multi-agent systems. Specifically, we focus on the challenges involved in adapting (originally disembodied) machine learning techniques to situated agents, the relationship between learning and communication, learning to collaborate and compete, learning of roles, evolution and natural selection, and distributed learning. In the second part of the chapter, we focus on some practicalities and present two case studies.

INTRODUCTION

Intelligence implies a certain degree of autonomy, which in turn, requires the ability to make independent decisions. Truly intelligent agents have to be provided with the appropriate tools to make such decisions. In most dynamic domains, a designer cannot possibly foresee all situations that an agent might encounter, and therefore, the agent needs the ability to learn from and adapt to new environments. This is especially valid for multi-agent systems, where complexity increases with the number of agents acting in the environment. For these reasons, machine learning is an important technology to be considered by designers of intelligent agents and multi-agent systems.

The goal of this chapter is not to present a comprehensive review of the research on learning agents (see Sen & Weiss, 1999, for that purpose) but rather to discuss important issues and give the reader some practical advice in designing learning agents.

The organization of the chapter is as follows. In the following section, the differences between pure machine learning and that performed by (single) learning agents are discussed. We start with the introduction of basic machine learning concepts, followed by examples of machine learning techniques that have been applied to learning agents, such as Q-learning, explanation-based learning, and inductive logic programming. In the third section, we discuss several issues surrounding multi-agent learning, namely, the relationship between learning and communication; learning to collaborate and compete; the learning of roles, evolution, and natural selection; and distributed inductive learning. Following this discussion, we focus on some practicalities and present two case studies. We finish the chapter with conclusions and further work.

FROM MACHINE LEARNING TO LEARNING AGENTS

In this section, we discuss the nature of machine learning (ML), its integration into agents, and the parallels between machine learning systems and learning agents. We start with a basic introduction to machine learning.

While most of the fundamental ML concepts introduced below are commonly associated with *supervised learning (SL)* (i.e., the generalization from annotated examples provided by a teacher), they are equally relevant for *reinforcement learning (RL)*, where an agent learns through the feedback (i.e., reinforcement) from the environment in each entered state. To date, most attention in agent learning has been reserved for RL techniques such as Q-

learning (see below), due to its suitability to situated agents. Nevertheless, we see SL and RL as strongly related approaches. In the case of RL, the environment could be seen as the teacher, and the generalization process would be over states (which correspond to the examples). In fact, SL methods can be directly applied in a RL setting (see, e.g., the use of neural networks in TD learning (Tesauro, 1992).

Note that we mostly exclude a third class of ML techniques, *unsupervised learning* (or *learning by discovery*), from our discussion, because there are only few research results in this area to date.

Introduction to Machine Learning

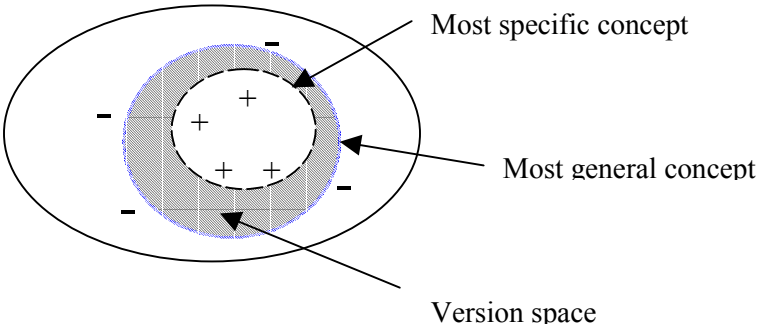
A computer program is said to learn *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E* (Mitchell, 1997). Machine learning can be seen as the task of taking a set of observations represented in a given *object* (or *data*) *language* and representing (the information in) that set in another language called the *concept* (or *hypothesis*) *language.* A side effect of this step can be the ability to deal with unseen observations. As an example, one can consider an object language that consists of pairs of coordinates (x,y). A number of observations are provided, and each is labeled as a positive or negative example of the *target concept*, i.e., the concept to be learned. Let the hypothesis language define a concept as an ellipse,[1] such that a point (x,y) would represent a positive example of the concept if it is inside that ellipse or a negative one otherwise. In the example in Figure 1, there are infinitely many such ellipses, each of which would satisfy the usual requirements for *completeness* and *consistency*, i.e., it would cover all positive examples and none of the negative. The set of all complete and consistent hypotheses for the definition of the target concept is referred to as the *version space* for that concept.

Selecting the Agent's Learning Bias

The hypothesis language specifies the *language bias*, which limits the set of all hypotheses that will be considered by the learner. In the example in Figure 1, it is obvious that there are certain arrangements of positive and negative examples on the plane that can never be separated by an ellipse in the required way.

A priori, all concepts in the version space are equally good candidates for the description of the *training* data provided. Only when additional *test* data is available can one find that certain concepts in the version space classify

Figure 1: Object and concept language.



unseen members of the target concept better than others. A *preference bias* is a principle that allows the learner to choose between two hypotheses if they both classify the training data equally. If hypotheses from the entire version space are to be ranked according to the preference bias, the order in which they are explored is of no importance. Alternatively, a *search bias* can be introduced to specify that order, and the search can stop after the first acceptable hypothesis with respect to a given criterion is found.

The appropriate choice of language, preference, and search biases is crucial for the learning outcome. The language bias has to be sufficiently general to include the target concept, yet restricted enough to make the search feasible. A well-chosen search bias can significantly speed the learning. At the same time, specifying the way in which the hypothesis space is searched can result in a preference bias being enforced, as in the cases when search starts with the most general, or the most concise, hypotheses. The choice of preference bias can be guided by some intuition about the type of hypothesis likely to be successful for the given type of data, or by some general principle, such as *Occam's razor*, favoring the hypothesis using the minimum number of entities, i.e., the simplest or shortest one, and the Minimal Description Length (MDL) principle, which recommends "*the shortest method for re-encoding the training data, where we count both the size of the hypothesis and any additional cost of encoding the data given this hypothesis*" (Mitchell, 1997).

Selecting the Agent's Learning Technique

Machine learning techniques can be classified according to a number of criteria. *Inductive learning* has the power to produce theories that are more general than the initial information provided, whereas *deductive learners* only specialize, i.e., restrict the coverage of existing theories. *White box* learners generate hypotheses that can be interpreted — and modified — by the user; the opposite is true for *black box* learners, which can be used if understanding of or changes in the hypothesis learned are not necessary. *Incremental learning* updates the current hypothesis when provided with additional examples, whereas *batch learning* has to process the whole data set again. Both can be used in an agent, but time constraints and quality of learning have to be taken into account, and the benefits of each method must be compared. *Eager learning* generates a hypothesis from the training data, which is then thrown away, and the hypothesis used unchanged for all test examples; for each test example supplied, *lazy learning* generates a separate hypothesis. One can see that eager learning saves memory, but, in comparison with lazy learning, requires more extensive computation to be done before test examples are supplied.

As mentioned above, the data used for learning can belong to one of three classes. *Annotated data* assigns to each observation a label, which can contain the name of the concept or some other information associated with the observation. In this case, known as *supervised learning*, the learning task is to find a hypothesis that, given the observation, will produce the label. *Unsupervised learning* deals with nonannotated data. Here, no explicit target concepts are given, and the aim of learning is to change the data representation in a way that highlights regularities in the data. Finally, in *reinforcement learning*, only some of the training examples are annotated (with the associated immediate rewards), and the learner has to propose the annotation of the remaining ones. From another point of view, one could also argue that reinforcement learning is a specific case of unsupervised learning, because the optimal action with respect to the agent's goal (e.g., maximum discounted cumulative reward) and time horizon (often infinite) is not explicitly given.

Parallels between Machine Learning and Learning Agents

Machine learning has gone through different stages with respect to the way in which training data is selected. In the classic setting, the ML algorithm does not choose the training data, which is provided by the user. *Active learning*

(Thompson et al., 1999) aims at minimizing the need of hand-annotated data. It starts with a training data set of which only a small part is annotated, and then it gradually requests manual annotation for those remaining training examples likely to be the most helpful for the learning process. *Close loop machine learning* (*CLML*) (Bryant & Muggleton, 2000) eliminates the need of an external annotator, and instead, autonomously plans and performs experiments based on its current hypothesis in order to obtain additional training data when necessary.

If we were to compare machine learning and learning agents, CLML would be considered a borderline case. CLML is, indeed, a ML technique, the implementation of which could also be seen as an autonomous, rational, and deliberative agent, with the single aim of learning. In the general case of learning agents though, learning is just one of many goals that an agent has to achieve within its limited resources, and additional issues, such as time constraints, have to be taken into account.

The remaining part of this section will introduce a few learning algorithms of specific relevance to learning agents.

Q-Learning

Q-learning, a reinforcement learning technique, is the most widely used learning method for agent systems. The goal of Q-learning is to compute a state-to-action mapping (a so-called *policy*) that leads to the maximum utility for the agent. The policy generation is based on a numerical reward feedback from the environment after each action execution. In other words, the agent learns by exploring the environment, i.e., experimenting with different actions and observing the resulting reward. The result of Q-learning, i.e., the policy, may be seen as a table that assigns each state–action pair (`s`,`a`) a numerical value, which is an estimate of the (possibly long-term) reward to be received when executing `a` in `s`. After receiving a reward, an agent updates the numerical value of the state–action pair based on the reward and on the estimated best reward to be gained in the new state. Thus, with time, the agent is able to improve its estimates of the rewards to be received for all state–action pairs. Due to space restrictions, we do not present details of the Q-learning algorithm but rather refer the reader to Mitchell (1997) or Kaebling et al. (1996).

The advantages of Q-learning are its efficiency, guaranteed convergence toward the optimum, and natural applicability to agent systems because of the coupling of learning and exploration. But even though most research on learning agents has been concentrated on Q-learning, it has a few drawbacks: (1) defining a suitable numerical reward function can be a nontrivial task for some

application domains; (2) convergence to the optimal (i.e., utility-maximizing) policy requires that all state–action pairs be visited infinitely often, which obviously leads to problems with large state–action spaces in real-world domains (note that the choice of exploration strategy influences the convergence speed); and (3) the learning result (a table assigning each state–action pair a numerical value reflecting the expected reward) is not transparent in the sense that no explanation is given for action preferences. Nevertheless, given a state, actions can be ranked, which may yield some limited insight into the performance of the agent.

Explanation-Based Learning

Computing a solution (e.g., a plan) directly from basic principles is a hard problem in terms of complexity. Explanation-based learning (EBL) adds potentially useful macro-operators to the knowledge base and thus speeds the search process for an optimal solution.

EBL has been widely used in artificial intelligence to speed the performance of planners [e.g, in Prodigy (Carbonell et al., 1990)]. Generally speaking, the agents are concerned with improving the efficiency of the problem solver rather than acquiring new knowledge. Obviously, problem solvers, when presented with the same problem repeatedly, should not solve it in the same way and amount of time. On the contrary, it seems sensible to use general knowledge to analyze, or *explain*, each problem-solving instance in order to optimize future performance. This learning is not merely a way of making a program run faster but also of producing a more accurate hypothesis.

In short, EBL extracts general rules from single examples by generating an explanation for the system's success or failure and generalizing it. This provides a deductive (rather than statistical) method to turn first-principles knowledge into useful, efficient, special-purpose expertise. The learned rules enable the planner to make the right choice when a similar situation arises during subsequent problem solving.

A practical example in a conflict simulation domain is presented in Section 4.

Inductive Logic Programming

Inductive logic programming (ILP) is a white-box learning method based on the use of induction, logic programming, and background knowledge (Muggleton & De Raedt, 1994). In more detail, ILP uses a subset of first-order predicate logic known as Horn clauses as its object and concept language. The object language is at times further restricted to ground facts, i.e., to a subset of

Table 1: ILP Object Language Example

Good bargain cars				***ILP representation***
Model	Mileage	Price	yes/no	gbc(#model,+mileage,+price).
BMW Z3	50000	£5000	yes	gbc(z3,50000,5000).
Audi V8	30000	£4000	yes	gbc(v8,30000,4000).
Fiat Uno	90000	£3000	no	gbc(uno,90000,3000).

propositional logic, in which case, the examples of the target predicate have a simple relational representation (see Table 1).

The concept language in this example could be restricted to the basic relations provided with the ILP learner, e.g.:

```
equal(X,X).

greater(X,Y) :- X > Y.
```

The language of Horn clauses is more powerful than propositional logic, as the latter, for instance, has to express the concept of equality between the values of two arguments as a list of pairs of values:

```
(arg1=1 & arg2=1) or (arg1=2 & arg2=2) ...
```

which is inefficient in the case of finite domains and impossible otherwise. In most ILP systems, one can learn one target concept at a time, but there are some notable exceptions (Progol5). Each of the attributes of the target concept has to be defined by its *type*, defining the range of values, and *mode*, describing whether in the concept definition that attribute will be an input variable, an output variable, or a constant (see last column of Table 1).

Apart from the standard, built-in concepts, the concept language in ILP can be extended with user-defined concepts or *background knowledge*. The use of certain background predicates may be a necessary condition for learning the right hypothesis. On the other hand, redundant or irrelevant background knowledge slows the learning. To continue with our example, one may consider helpful a concept that associates with a car the product of its mileage and price

and compares it with a fixed threshold T. In logic programming speak, this can be represented as:

```
prod(Miles, Price, T) :- Miles * Price < T.
```

Now, one of the possible definitions of a good bargain BMW Z3 can be expressed as a car of that model for which the product mentioned is below 250000001:

```
gbc(z3, Miles, Price) :- prod(Miles, Price, 250000001).
```

The preference bias in ILP is typically a trade-off between the generality and the complexity of the hypothesis learned; some ILP learners allow the users to specify their own bias.

MACHINE LEARNING FOR MULTI-AGENT SYSTEMS

Learning becomes a much more complex task when moving from a single agent to a multi-agent setting. The environment becomes more dynamic and less predictable due to many (potentially adversarial) agents acting in it, each equipped with its own goals and beliefs. New issues arise such as coordination in teams of cooperating agents and competition with adversarial agents, all of which can (and should) be improved by learning.

In this section, we present several important issues concerning the application of ML in a multi-agent setting. First, we contrast the straightforward approach of directly transferring single-agent learners into a multi-agent domain with the more complex approach of designing learning agents with a social awareness. We continue with a discussion of two major learning goals, namely, learning to coordinate and learning to compete. We then present issues surrounding communication, team roles, and evolution for learning, all of which are highly relevant in a multi-agent domain. Finally, we take a brief look at distributed learning and its application to data mining.

The area of multi-agent learning is still young, and therefore, many of the questions we present have no definitive answer yet. Nevertheless, they are important considerations for the designers of learning agents.

Multiple Single-Agent Learning Versus Social Multi-Agent Learning

An obvious question is why not use the same single-agent learning techniques discussed in the previous section directly (i.e., without further modification) in a multi-agent setting, an approach we call *Multiple Single-Agent Learning*. In that case, a learning agent would perceive other agents only as a part of the environment and have no explicit awareness of their existence, let alone their goals and beliefs. Nevertheless, changes in the environment due to the actions of other agents are still being perceived, and thus, a model of other agents can be acquired indirectly during the learning process as part of the environment model.

In contrast, agents can have a high awareness of other agents and incorporate this knowledge in the learning process, potentially using communication, coordination, and agent modeling techniques to support the learning task. While this *Social Multi-agent Learning* approach is certainly more complex, does it necessarily lead to improved learning performance?

First, it is interesting to note that for nonlearning agents, social awareness is not necessary to achieve near-optimal behavior, as has been shown in experiments in a simple foraging domain (Steels, 1990). For learning agents, initial experiments in a simple two-player cooperative game setting (Claus & Boutillier, 1998; Mundhe & Sen, 2000) show the surprising results that social awareness is not always beneficial in learning and may even hurt performance under certain conditions. The research draws on a classification of different levels of social awareness (Vidal & Durfee, 1997): level-0 agents have no knowledge and awareness of other agents, while level-*k* agents model other agents as having level at most (*k-1*). For example, a level-1 agent uses a model of the other agents, but this model assumes that other agents are of level 0, i.e., their actions are not influenced by any model of the level-1 agent (i.e., they are not choosing their actions based on direct observations of the agent's past actions). Mundhe and Sen looked at the behavior of Q-learning agents of social awareness levels 0, 1, and 2 and the impact of these levels on convergence in terms of speed and effectiveness of the result. Their experiments show that two level-1 agents display the slowest and least effective learning, worse than two level-0 agents. While these results are a first indication that social awareness may decrease learning performance, the experiments have been carried out in a rather simple setting, and there is the need for further research into these issues in order to be able to give more general guidelines to the agent designer.

Learning to Coordinate and to Compete

Characteristically, agents must learn to collaborate and to compete in multi-agent domains. Not only do the players have to learn low-level skills, but they must also learn to work together and to adapt to the behaviors of different opponents.

Team games are a rich domain for the study of collaborative and adversarial multi-agent learning. Teams of players must work together to achieve their common goal, while at the same time defending against the opposing team. Learning is essential for this task, because the dynamics of the system change due to changes in the opponents' behaviors.

Stone and Veloso (1998) illustrate how agents learn to cooperate and compete in a robotic soccer scenario: The passer in (robotic) soccer would need to pass the ball in such a way that the shooter could have a good chance of scoring a goal. The parameters to be learned by the passer and the shooter are the point at which to aim the pass and the point at which to position itself, respectively.

At the same time, as teammates are cooperating and passing the ball among themselves, they must also consider how best to defeat their opponents. As time goes on, the opponents need to co-evolve in order to adjust to each other's changing strategies.

Reinforcement Learning for Multi-Agent Systems

In most approaches to multi-agent learning, reinforcement learning (specifically Q-learning) has been the method of choice for the same reasons as in a single-agent setting (see also the previous section). While reinforcement learning can usually be applied to MAS straightforwardly in a multiple single-agent learning setting (e.g., Crites & Barto, 1998), problems arise when agents working in a team are not able to observe each others' actions all of the time. Cooperating agents often receive rewards globally as a team for the combined actions rather than locally for individual actions. When updating the Q table based on such a reward and in the absence of information about the other agents' actions, an agent has to use an heuristic to determine the contribution of each agent to this reward. Lauer and Riedmiller (2000) present two such heuristics:

- *Pessimistic assumption:* Given my action, the other agents always perform actions that yield the minimum payoff for the team.
- *Optimistic assumption:* Given my action, the other agents always perform actions that yield the maximum payoff for the team.

If each agent uses the pessimistic assumption, then this leads to overly cautious behavior and thus to slow convergence toward the optimal policy (without guarantee of convergence). Lauer and Riedmiller show that an optimistic assumption for each agent leads to guaranteed convergence. Nevertheless, no results have been presented on convergence speed.

Note that Lauer and Riedmiller's approach does not require any communication among the agents. We look at this and the resulting improvements in learning performance in the following subsection.

Learning and Communication

When several learning agents work in a team, it may be beneficial for them to cooperate not just on the task achievement but also on the learning process. Clearly, communication is an important tool for such cooperation. Tan (1993) considered the following types of communication between Q-learning agents in the hunter–prey domain:

- *Sharing sensation:* Each agent has only a limited sensory range that can be extended by each agent communicating its sensory information to the others.
- *Sharing policies:* All agents share the same policy in the form of a blackboard. Each agent is permitted to update the policy based on its experience.
- *Merging policies:* After a certain number of moves, the policies of two or more agents are merged, e.g., by averaging the values in the Q tables.
- *Sharing episodes:* After a successful completion of an episode, the corresponding action–reward sequence is communicated to all other agents, who use it as a training episode.

As expected, empirical results show that communication during Q-learning speeds up the convergence. Nevertheless, in some cases, sharing sensory information can detract an agent's learning from the correct hypothesis and hurt the learning process.

In related work by Provost and Hennessy (1996), a team of inductive rule learners generate a joint hypothesis. The individual learners communicate the individually computed classification rules, and the other agents critique them based on their local training data. Only those rules that receive a positive feedback are used in the final hypothesis.

The above examples show the use of low-level communication that is mainly an exchange of data. To date, higher-level communication, such as

indications of intentions or negotiation of learning targets has not been looked at in depth in the context of multi-agent learning. We are currently developing role learning techniques that incorporate an explicit communication and negotiation for the coordination of learning in a team.

Learning of Roles

When working in a team, specialization of team members on specific tasks or task categories may be beneficial. One way to achieve this kind of team heterogeneity in a MAS is to equip agents with different behaviors or sensor and effector capabilities and thus predefine the roles that they are going to play in the team. While this method may lead to good results (Parker, 1994), it has a number of drawbacks. First, it is not always obvious how to specify an optimal (or even useful) distribution of behaviors. Second, it may be quite expensive (in terms of hardware or in terms of development time) to design a system of heterogeneous agents.

An alternative is to use a team of learning agents that are homogeneous to begin with but with time and experience will diversify and specialize. There are two main questions that a learning MAS designer faces: (1) How can agent specialization be achieved by learning, and (2) does the application benefit from such team heterogeneity?

While as yet there are no definitive answers to these questions, research results are available that shed some light on them. Prasad et al. (1996) present a method to learn the optimal distribution of predefined roles (i.e., behaviors) in a team. While this is not role learning per se, it is a first noteworthy approach to answer the above questions.

Balch (1999) studied the conditions under which a team of agents based on reinforcement learning will converge toward heterogeneity. In his research, he distinguishes between two main types of reward functions: *local*, where each agent receives rewards individually for personally achieving a task; and *global*, in which all team members receive a reward when one of the team members achieves a task. Empirical results show that globally, reinforced agents converge toward a heterogeneous team, while local reinforcement leads to homogeneous agents. Furthermore, learning heterogeneity is not always desirable: in multirobot foraging, a locally reinforced and therefore homogeneous team outperforms a globally reinforced and therefore heterogeneous team. On the other hand, in the robotic soccer domain, Balch's results are the opposite, i.e., global reinforcement (and thus heterogeneity) yields the better performance.

In related work, Crites and Barto (1998) show that a team of elevators that are locally reinforced do not display heterogeneous behavior, but they nevertheless perform highly effectively. Tumer and Wolpert (2000) discuss the connection of local rewards and the total world utility. Their wonderful life utility (WLU) permits agents to remove the noise and uncertainty related to the activity of other agents and focus on how the agent contributes to the world utility. This WLU has been shown to result in higher world utilities when used in teams of RL agents (as compared to global rewards).

While these results do not provide a general answer to the when and how of learning roles, they show that team heterogeneity is an important factor in multi-agent performance, and that different learning methods can yield different levels of heterogeneity. It is an interesting open research problem to gain a clearer and more detailed understanding of the relationship between agent diversity and performance and learning in general.

Natural Selection, Language, and Learning

The task of any learning agent is to modify the agent's knowledge under the guidance of some fixed metaknowledge contained in the language, preference, and search biases. Introducing several metalevels does not change the fact that the top one will still have to be fixed and provided from a source external to the learner. That source can be the agent's creator, but the bias can also be the product of evolution. In the latter case, the bias will be set at random in a population of agents, in which the principles of natural selection are applied, so that the "fitter" the agent, i.e., the better it achieves its goals, the higher the chance that some aspects of its bias will be used in the new agents introduced in the environment. The exact way in which these aspects are propagated into the agent's "offspring" can be modeled with the standard notions of genetic crossover and mutation, so that the bias of every new agent is selected as a combination of the suitably represented biases of its two parents, to which small random changes are introduced.

Whether knowledge and skills learned by an individual can influence the inherited features it will pass on to the offspring, has been the subject of many theories. According to Darwin, individual experience cannot change one's genes; Lamarck believed the opposite possible. While Darwin's theory has been universally accepted to reflect the true mechanisms of inheritance in nature, there are no reasons why one should not experiment with Lamarckian evolution among artifacts. The way in which Darwinian evolution is related to learning is captured in the so-called Baldwin effect (Baldwin, 1896). The effect

predicts that the evolution of new behavior goes through two phases. Initially, learning is favored by natural selection, as it helps to patch the inherited behavior, while the latter is still incomplete or suboptimal. In stable environments, as inherited behavior gets closer to the optimal, it gradually displaces learning to avoid the cost associated with the latter. One can represent the overall trend as a tendency to strengthen the language bias, so that search space is pruned to eliminate bad hypotheses and zoom in on the likely winners (Turney, 1996).

From a MAS designer's point of view, Lamarckian evolution is faster but brings the risks of inheriting too-specific concepts based on the parents' personal experiences that have no analogue in the life of the offspring. One could expect from Darwinian evolution to encode as inherited only general concepts, as they would have to have remained for *many* generations relevant to a substantial part of the population. There is, however, a third way open to populations of agents able to communicate. Language uses concepts that are specific enough to be useful in the description of a variety of aspects of the agent's environment (including other agents), yet general enough to correspond to shared experience. In this way, the concepts of a shared language serve as a language bias, which is inherited by the individuals through upbringing rather than genes. To preserve the additional advantage that the use of language brings about in the case of a changing environment, one should allow the language to evolve along with all other inherited features.

Distributed Inductive Learning

When learning over large or physically distributed training data sets, using just one agent to compute a hypothesis is no longer feasible. In such cases, several learning agents, each receiving a subset of the training data, are needed to compute a global hypothesis for the complete training data set.

Recently, many application areas for this kind of distributed inductive learning have emerged. First and foremost, distributed learning is a highly effective way of solving data mining problems caused by increasing data set sizes of physically distributed data (e.g., on the Internet) (Provost & Kolluri, 1999). In robotic soccer, player agents have only limited local observations available in order to induce global properties of the game, e.g., the strategy of the opposing team (we are currently working on distributed inductive learning algorithms that achieve this task).

Current distributed inductive learning methods are employing one of two general approaches:

- *Hypothesis combination:* Each agent individually computes a local hypothesis based on the local training data that are then combined into a global hypothesis, by a separate agent (e.g., Fayyad et al., 1993) or collaboratively by the local agents (e.g., Provost & Hennessy, 1996).
- *Hypothesis update:* One agent starts by inducing a local hypothesis from the local training data and then communicating it to another agent, which updates its hypothesis based on its own local data. An instance of this incremental batch learning approach can be found, e.g., in Domingos (1996).

There is still plenty of room for further research in distributed inductive learning, specifically in the area of explicit coordination during the learning process (which is related to research discussed in the subsection on communication).

INTEGRATION OF ML INTO MULTI-AGENT SYSTEMS

General Issues

As discussed in Section 2, the application of ML to agents involves many issues. Whereas the most important issue for a stand-alone machine learning algorithm is the quality of the theory learned, in a learning agent, the time needed for learning becomes a primary factor. The fact that computational resources of an agent have to simultaneously maintain several tasks, such as perception, planning, and control, imposes time constraints on learning. These constraints are relatively flexible, because individual tasks can be rescheduled, as long as the hard constraints imposed by the environment are met (find food or die, run or get eaten, etc.). Time constraints can be a significant factor in the choice of the learning strategy. In comparison to lazy learning, eager learning typically results in more compact theories, which are faster to use, but take more time to learn. A combination of both methods is also possible, so that observations are minimally preprocessed on the fly, and the computationally expensive part of the learning is postponed until sufficient time is available. For instance, certain hypothesis languages, such as first-order decision lists (Kazakov & Manandhar, 2001) permit all new observations that are not handled correctly by the current hypothesis to simply be added to a list of exceptions, as a lazy learner would do. At a later stage, an eager learning algorithm can be used to replace that list with a more compact hypothesis, if at all possible.

Apart from the time complexity and accuracy of the results, another consideration in choosing a particular learning algorithm is whether its worst-case complexity is known or not. For learning algorithms that explore the entire hypothesis, space finding an optimal solution is guaranteed after running a prescribed number of steps. Other algorithms, sometimes known as any-time algorithms, cannot recognize an optimal solution when they see one but can only compare the relative merits of two candidates. As a result, a longer run always has the potential to improve the best solution found so far. Using the former type of algorithms means that one can use worst-case complexity analysis to find an upper bound on the execution time and allocate time for learning accordingly. This is not possible with any-time algorithms. Nevertheless, they have the advantage of being able to provide at any time a draft of the hypothesis being learned. A possible policy in this case is to stop learning to meet deadlines or when cost outweighs expected improvements of accuracy.

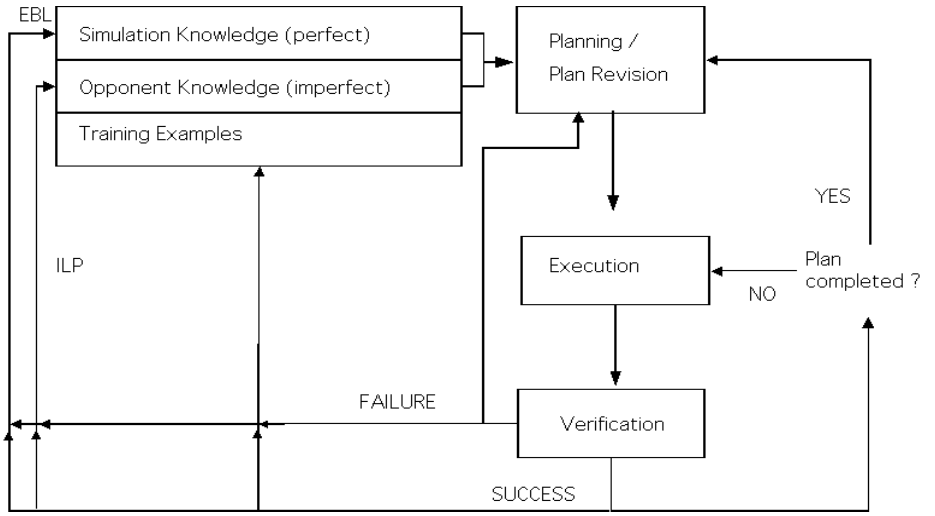
MAL for Conflict Simulations

Our first case study involves a conflict simulation (CS) as a particular example of game playing. Conflict simulations provide a suitable and challenging application domain with which to test and evaluate logic-based learning techniques such as EBL and ILP, mainly because a large amount of useful background knowledge is readily available in the form of the simulation model. RL has been widely applied to simple games such as Robocup (e.g., Tambe et al., 1999) where domain knowledge is not necessary for the agents to learn and perform well.

A command and control hierarchy was chosen as a natural organizational structure for the CS domain. The main target concept is "being successful" (i.e., eliminate the enemy units). However, different agents will learn such a concept differently, according to their roles and corresponding knowledge bases. Commanders will use abstract knowledge to form and learn strategic and operational plans, whereas particular units will use domain-specific knowledge to form and learn tactical plans. Each agent in the hierarchy learns on its own how to improve its skills. Units learn how to succeed in moving and attacking the enemy. Commanders, whose skills and action repertoire are limited to issuing orders, will learn how to have their sections effectively coordinated. Coordination is a matter of command. Consequently, units do not learn how to cooperate—commanders do.

Some might argue that this is a type of *isolated* learning (Stone & Veloso, 1998), because agents seem to learn separately and individually. However, what a unit learns will affect the planning and learning processes of other agents.

Figure 2: A CS agent architecture.



If a unit learns how to achieve its goal effectively, its commander will assign it the same task in a similar operation. If the unit learns that it cannot achieve the goal, it will be replaced, or the original commander's plan will change accordingly. As a consequence, new assignments (roles) or combinations of assignments will be learned.

The agent architecture and how it learns is depicted in Figure 2. A CS agent has three different knowledge bases: one containing perfect information about the rules of the simulation, one containing induced knowledge about the opponent's behavior, and one in which a history of past successes and failures is stored to serve as training examples for learning. After an action has been executed and the result observed, it can be potentially used as a training example for the machine learning algorithms.

The learning result is added to the respective knowledge base: results of explanation-based learning are added to the perfect knowledge base (given that the generalization process was based on perfect knowledge), and the results of ILP learning are added to the imperfect knowledge base, because there is no guarantee of correctness for inductive reasoning.

We sketch how EBL works with a simple example. Let us assume that a commander has sent its units to eliminate an enemy unit and that these units have chosen their locations in such a way that the enemy is surrounded. Let us further

assume that they are successful and achieve their goal. In order to explain this success, the commander then uses the following rules that were not used previously (although they have always been in the knowledge database):

Fact: Each unit has a zone of control, i.e., the six hexagons adjacent to its current location.
Rule 1: A unit entering an enemy zone of control must stop its movement phase.
Rule 2: A unit withdrawing into an enemy zone of control is eliminated.

With these rules, it is now easy for the commander to explain the success of its section. They succeeded because they surrounded the enemy unit, cutting all its possible paths of withdrawal. This rule is then generalized:

EBL Rule: If a unit is surrounded, then it is eliminated.

This new general rule is then added to the knowledge database and used directly to elaborate more precise plans more quickly. Of course, nothing new was learned. Theoretically, the agents could deduce such a rule from the original database. However, it would have been a more expensive process to do so each time it was needed.

It is one thing to deduce from general rules the order and assignment of the subtasks that should, in theory, lead an army to victory; it is another to learn from examples (and some prior knowledge) whether the conditions for such subtasks to be executed successfully actually hold. In contrast to EBL methods, ILP computes a hypothesis not just based on simulation rules known beforehand but also on external and initially unknown circumstances, such as opponent's strategies. Generally, relying exclusively on EBL-generated rules can turn out to be impractical in real-world domains in which agents work with incomplete knowledge, and thus, ILP is an important addition to the system's effectiveness.

We illustrate the use of ILP in our system with an example. An agent might deduce the following rule from the current knowledge base (taking into account, for example, the units' combat strength).

Rule: If a single tank unit attacks an infantry unit on a bridge, it is successful.

The unit uses this rule to compute and execute an appropriate plan (move to the adjacent location, etc.). The plan, however, fails repeatedly (e.g., the infantry unit withdraws and blows the bridge up). These new examples

contradict the above rule. ILP will be used to find a new hypothesis. In this particular case, the learned hypothesis is the negation of the previous rule.

Hypothesis: If a single tank unit attacks an infantry unit on a bridge, it is not successful.

The unit will then rule out a frontal attack and try to find an alternative plan (e.g., moving some friendly units to the other side of the bridge first).

The implementation of the CS consists of two parts. The user interface (user input, map/agent visualization, plan/move visualization, communication visualization, and agent hierarchy visualization) is being implemented in JAVA. The simulation server and the agents are being implemented in Progol. The simulation is synchronized and turn-based. In other words, all agents of one side send their actions to be executed simultaneously to the simulation server, which then computes a world-update accordingly and sends the changes back to all agents. Afterwards, the agents of the opposing side take their actions, and so on.

Communication could be implemented in MAS-Progol, a multi-agent version of Progol that is being developed in the AI research group at the University of York. It enables an agent to send Prolog goals to other agents and receive variable bindings or truth values in return.

More details on the multi-agent system architecture and implementation can be found in Alonso & Kudenko (1999) and Kudenko & Alonso (2001).

The York Multi-Agent Environment

The York MAE has been developed as a general-purpose platform on which to study the integration of various machine learning techniques in agents, with a focus on ILP and evolutionary techniques. The platform allows the user to define a simulated ecosystem with different types of terrain and a number of artificial species. Animal-like species are defined by the types of terrain they can walk over, the species they feed on, and the ones that prey on them; one can further define the senses that each species possesses, their range, and the medium used (e.g., light). There is also one predefined type of plant to start the food chain. The behavior of an individual is defined by three factors:

- *Default behavior:* This is a relatively simple behavior that has to be coded directly at source level and is shared by all individuals of a species. The default behavior is defined as a range of drives related to elements of the environment that are necessary for survival: thirst, hunger, sex drive, fear.

Each time the agent has to take an action, all drives are evaluated, and the strongest triggers an action aiming at its reduction.

- *Inherited features:* The default behavior can be parametrized, and those parameters are subjected to natural selection. Each agent has an array of integers used to store the inherited features. Under certain conditions, two agents can mate and produce offspring, with genetic material obtained from the parents. The genetic operators of crossover and mutation are implemented as standard methods and can be used with minimum effort.
- *Learning:* Each agent can be optionally allocated a separately running process of the ILP learner Progol (Muggleton & Firth, 2001). The agent then can send observations to the learner and receive recommendations about its next action.

The above setting permits the use of evolution and individual learning separately or in conjunction, so that, for instance, Darwinian evolution of the ILP language bias is combined with ILP learning in individuals. The discrete coordinate system is suitable for experiments with Q-learning, which can be easily implemented in Prolog, and run on Progol, as the latter contains a full Prolog interpreter.

The implementation of the system involved a number of issues related to learning. For instance, the designers had to decide how to implement individual agents. Having them as separate threads or processes would lead to the most realistic simulation, where agents would act and learn asynchronously. That would also put the strongest pressure on the learning component of an agent's behavior. Because most machine learning techniques have not been developed with real-time use in mind, it was deemed more reasonable to opt for a different setting, where time pressure on learning could be controlled and increased gradually. That led to a design in which each agent is prompted in turn to select an action. The initial intuition is that carrying out all agents' actions simultaneously would mean that no agent is favored. However, mutually exclusive actions would lead to conflicts. To resolve the conflicts, one would have to employ ranking among agents, and favor the ones with the higher rank. Instead, the implementation was based on a loop that rearranges all agents at random then prompts each to choose an action that is carried out immediately. In this setting, if the agent's decision process involves learning, it can take all the time it needs, while ensuring that the environment and the other agents remain the same. Time pressure can be added gradually by imposing a limit on the time used to learn or, more generally, to make a decision.

As each agent's learning is implemented as a separate process, one can use several processors or computers to implement true parallelism. This opens the door for several strategies combining learning with the use of default behavior. One could wait for the actions suggested by the two and go for the more conservative, or wait for the learning component for a limited period of time and then, if no answer is obtained, use the (easily computed) default behavior. If the behavior based on learning is considerably more successful than the default, but also slower, one may decide to use the actions proposed by the former, even if their choice has not been based on the most recent observations. These, again, can be compared to the actions suggested by the default behavior on the basis of the latest data, and a trade-off can be chosen.

The York MAE has been developed as part of several undergraduate student projects. It has been used so far for simulations of kinship-driven altruism. The system is available on request for noncommercial purposes.

SUMMARY AND CONCLUSION

The ability to learn is a central feature of intelligent agents. In this chapter, we presented many issues surrounding the application of machine learning to agents and multi-agent systems. Furthermore, we presented two examples of systems that are being developed at the University of York.

ML for agents is still a relatively young area, and there are many issues that still need further research, some of which have already been mentioned. While many agent and multi-agent learning techniques and methods still need more evaluation to prove their practicality, we summarize a number of general recommendations for learning agent designers:

- Worst-case and average time complexity of the learning algorithm and the theory it produces should be considered when a learning agent is being developed. Also, provisions may be made to ensure that the agent has a partial theory or a fallback plan to act upon should learning or recall have to be interrupted.
- The choice of bias can prove crucial to the speed and accuracy of an agent's learning. In the cases where this choice is not obvious or in dynamic environments, a good engineering approach can be to employ natural selection among agents for the search of the best bias.
- When designing learning agents for multi-agent systems, try multiple single-agent learning first. This is much simpler and may already lead to good results. Should it fail, gradually increase the social awareness of the agents.

- While Q-learning is the most common method to date, it may cause difficulties in more complex real-world domains. It is worth looking at alternative methods, especially hybrid solutions (e.g., Dzeroski et al., 1998).
- Communication between learning agents can significantly improve the coordination and learning performance. Even simple forms of communication such as sharing experience have a significant effect.
- In some multi-agent applications, role heterogeneity is beneficial and in others, role homogeneity. When using RL approaches, one should take this into account in the choice of reward functions.

In addition to the open problems mentioned earlier in this chapter, further areas of future work on learning agents include:

- *Formal models of MAL:* To date, most developers are not able to predict the behavior of learning agents and depend purely on observing emergent patterns. Formal models of MAL that can be used to predict (or at least constrain) the behavior of learning agents would be useful. For example, Gordon's research (Gordon, 2000) describes one such approach.
- *More complex applications:* Most MAL application domains are relatively simple. It would be interesting to see MAL research for more complex, real-world applications. Eventually, such applications would encourage researchers to look beyond pure reinforcement (and specifically Q) learning, as suggested in the previous section.

ENDNOTES

1 An ellipse is described by a five-tuple: the coordinates of its two foci, and another constant representing the sum of the distances of each of the ellipse's points to the two foci.

2 Only vertical communication (communication among agents in two adjacent levels of the hierarchy) is allowed. Moreover, the content of the messages is also restricted to orders and requests.

BIBLIOGRAPHY

Alonso, E. & Kudenko, D. (1999). Machine learning techniques for adaptive logic-based multi-agent systems. *Proceedings of UKMAS-99*.

Balch, T. (1999). Reward and diversity in multi-robot foraging. *Proceedings*

of the IJCAI-99 Workshop on Agents Learning about, from, and with other Agents.

Baldwin, J.M. (1896). A new factor in evolution. *The American Naturalist, 30*.

Bryant, C.H. & Muggleton, S. (2000). Closed loop machine learning. *Technical Report YCS330*, University of York, Department of Computer Science, Heslington, York, UK.

Carbonell, J., Knoblock, C., & Minton, S. (1990). PRODIGY: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for Intelligence*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Claus, C. & Boutillier, C. (1998). The dynamics of reinforcement learning in cooperative multi-agent systems. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*.

Crites, R. & Barto, A. (1998). Elevator group control using multiple reinforcement learning agents. *Machine Learning, 33*.

Domingos, P. (1996). Efficient specific-to-general rule induction. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*.

Dzeroski, S., De Raedt, L., & Blockeel, H. (1998). Relational Reinforcement Learning. *Proceedings of the Eighth International Conference ILP-98*, Heidelberg: Springer-Verlag.

Fayyad, U., Weir, N., & Djorgovski, S. (1993). SKICAT: A machine learning system for automated cataloging of large scale sky surveys. *Proceedings of the Tenth International Conference on Machine Learning*.

Gordon, D. (2000). Asimovian Adaptive Agents. *Journal of Artificial Intelligence Research, 13*.

Kaebling, L.P., Littman, M.L., & Moore, A.W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research, 4*.

Kazakov, D. & Manandhar, S. (2001). Unsupervised learning of word segmentation rules with genetic algorithms and Inductive Logic Programming. *Machine Learning, 43*.

Kudenko. D. & Alonso, E. (2001). Machine learning for logic-based multi-agent systems, *Proceedings of the First Goddard Workshop on Formal Methods for Multi-agent Systems*, Springer LNAI. To appear.

Lauer, M. & Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. *Proceedings of the 17th International Conference in Machine Learning*.

Mitchell, T. (1997). *Machine Learning*. New York: McGraw-Hill.

Muggleton, S. & Firth, J. (2001). CProgol4.4: a tutorial introduction. In S.

Dzeroski & N. Lavrac (Eds.), *Relational Data Mining,* (pp. 160-188) Springer-Verlag.

Muggleton, S. & De Raedt, L. (1994). Inductive logic programming: theory and methods. *Journal of Logic Programming, 19*.

Mundhe, M. & Sen, S. (2000). Evaluating concurrent reinforcement learners. *Proceedings of the Fourth International Conference on Multi-agent Systems*, IEEE Press.

Parker, L.E. (1994). *Heterogeneous multi-robot cooperation*. PhD thesis, MIT Department of Electrical Engineering and Computer Science.

Prasad, M.V.N., Lander, S.E., & Lesser, V.R. (1996). Learning organizational roles for negotiated search. *International Journal of Human-Computer Studies, 48*.

Provost, F. & Hennessy, D. (1996). Scaling up: distributed machine learning with cooperation. *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*.

Provost, F. & Kolluri, V. (1999). A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery, 3*.

Sen, S. & Weiss, G. (1999). Learning in multi-agent systems. In G. Weiss (Ed.), *Multi-agent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press.

Steels, L. (1990). Cooperation between distributed agents through self-organization. In Demazeau, Y. & Mueller, J.P., (Eds.), *Decentralized AI — Proceedings of the First European Workshop on Modeling Autonomous Agents in a Multi-agent World (MAAMAW-89)*. Amsterdam, New York: Elsevier Science.

Stone, P. & Veloso, M. (1998). Towards collaborative and adversarial learning: a case study in robotic soccer. *International Journal of Human Computer Studies, 48*.

Tambe, M., Adibi, J., Alonaizon, Y., Erdem, A., Kaminka, G., Marsella, S., & Muslea, I. (1997). Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110.

Tan, M. (1993). Multi-agent reinforcement learning: independent versus cooperative agents. *Proceedings of the Tenth International Conference on Machine Learning*.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*.

Thompson, C., Califf, M.E., & Mooney, R. (1999). Active learning for natural language parsing and information extraction. *Proceedings of the 16th International Conference on Machine Learning*.

Tumer, K. & Wolpert, D. (2000). Collective Intelligence and Braess' Paradox. *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*.

Turney, P. (1996). How to shift bias: lessons from the Baldwin effect. *Evolutionary Computation*, *4*.

Vidal, J.M. & Durfee, E. (1997). Agents learning about agents: a framework and analysis. *Working Notes of the AAAI-97 Workshop on Multi-agent Learning*.